Resilient and Scalable Forwarding for Software-Defined Networks with P4-Programmable Switches

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Daniel Alexander Merling
aus Nürtingen

Tübingen
2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	19.04.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Michael Menth
2. Berichterstatter:	Prof. Dr. Wolfgang Küchlin

Contents

List of Abbreviations iii

Summary vii

List of Publications xi

1 Introduction & Overview 1
1.1 Software-Defined Networking and Data Plane Programming 1
1.2 Programmable Protocol-Independent Packet Processor (P4) 3
1.3 Bit Index Explicit Replication (BIER) 4
1.4 Research Objectives 5
1.5 Research Context 6
1.6 Research Results 6

2 Results & Discussion 9
2.1 Protection of Data Plane Traffic in SDN with P4 9
2.1.1 Robust LFA Protection for Software-Defined Networks (RoLPS) 10
2.1.2 P4 Protect 21
2.2 BIER-Based Multicast in P4 23
2.2.1 BIER Overview 24
2.2.2 BIER Fast Reroute (BIER-FRR) 25
2.2.3 BIER Scalability 25
2.2.4 BIER Implementation in P4 30
2.2.5 Discussion and Outlook 33
2.3 Additional Content 35
2.3.1 P4 ABC 35
2.3.2 Load Profile Negotiation 35

Personal Contribution 43

Publications 51
1 Accepted Manuscripts (Core Content) 51
1.1 Robust LFA Protection for Software-Defined Networks (RoLPS) 51
1.2 P4-Protect: 1+1 Path Protection for P4 70
1.3 An Overview of Bit Index Explicit Replication (BIER) 77
1.4 Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast 92

1.5 P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast 101
1.6 Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4 145
1.7 Efficiency of BIER Multicast in Large Networks 161
1.8 Learning Multicast Patterns for Efficient BIER Forwarding with P4 176
1.9 A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research 192
2 Accepted Manuscripts (Additional Content) 337
2.1 Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC) 337
2.2 Load Profile Negotiation in Day-Ahead Planning for Compliance with Power Limits 350

List of Abbreviations

ALD	advanced loop detection
API	application programming interface
BIER	Bit Index Explicit Replication
BIER-FRR	BIER fast reroute
BFD	bidirectional forwarding detection
BFIR	bit forwarding ingress router
BFR	bit forwarding router
BFER	bit forwarding egress router
CLI	command line interface
DLF	double link failure
eLFA	explicit LFA
FRR	fast reroute
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IPMC	IP multicast
LP	link protection
LFA	Loop-Free Alternate
MAT	match-action-table
MPLS	Multiprotocol Label Switching
NH	next-hop
NP	node protection
OF	OpenFlow
PLR	point of local repair
rLFA	remote LFA
RoLPS	Robust LFA Protection for Software-Defined Networks
SDN	software-defined networking
SLF	single link failure
SLF+SNF	single link failures plus single node failures
SNF	single node failure
TI-LFA	topology-independent LFA

Danksagung

Ganz besonderer Dank gilt meinem Doktorvater Prof. Dr. Michael Menth. Ihm habe ich die vielen Momente des Kopfzerbrechens, arbeitsreiche Wochenenden und meine Obsession für englische Kommaregeln zu verdanken. Aber noch viel wichtiger: Er gab mir die Möglichkeit mich selbst weiterzuentwickeln und an unzähligen Herausforderungen zu wachsen. Auch wenn seine hervorragende fachliche und wissenschaftliche Betreuung vermutlich den größten Einfluss auf meine Dissertation hatte, so schätze ich ihn doch vor allem für seine Persönlichkeit. Seine Art als Mensch und wie er den Lehrstuhl leitet haben dazu geführt, dass mir trotz aller Forschung, Diskussionen, Vorträge, Papern und Betreuung von Abschlussarbeiten immer genug Zeit für meine Familie und mich blieb. Deshalb nochmal direkt an Sie, Herr Menth: Vielen Dank!

Weiterhin möchte ich mich ausdrücklich bei meinem zweiten Gutachter Prof. Dr. Wolfgang Küchlin bedanken. Prof. Dr. Andreas Zell und Jun. Prof. Dr. Setareh Maghsudi danke ich für ihre Bereitschaft, als Prüfer an meiner Disputation teilzunehmen.

Alle Paper dieser Dissertation entstanden durch intensive Zusammenarbeit mit Kolleg:innen und Student:innen. Namentlich möchte ich mich bei Steffen Lindner, Marco Häberle, Frederik Hauser, Thomas Stüber, Erfan Mozaffariahrar, Lukas Osswald, Benjamin Steinert, Jonas Primbs, Manuel Eppler und Gabriel Paradzik bedanken. Ganz besonders möchte ich Gülsen Ergün-Karagkiozidou erwähnen und mich bei ihr bedanken. Sie ist für mich viel mehr eine manchmal große und manchmal kleine Schwester als nur eine Kollegin. Vielen Dank für dich als Mensch und deine Art! Ihr alle macht den Lehrstuhl zu dem was er ist. Ich hatte immer das Gefühl mehr mit Freunden zu arbeiten als mit Kollegen. Das nächste BIER [WRD+17] geht auf mich.

Zum Schluss aber nicht weniger wichtig möchte ich mich bei meiner Familie, meinen Freunden, Bekannten, Fast-Food Lieferanten, Zweckgemeinschaften und Spotify bedanken. Bei meinen Eltern, Eva und Herbert, die mit fast nichts nach Deutschland kamen und so viel gegeben haben, um mir so viel zu ermöglichen, die immer für mich da sind und meine Eigenheiten aushalten. Bei meinem kleinen Bruder Dominik, der

nie zu müde ist mich zu triezen. Bei meinen besten Freunden, Maxi, Max und Alex ohne deren unermüdliche Ablenkung mit Blödsinn meine Dissertation schon lange fertig gewesen wäre. Und bei meiner Freundin Lelly, die so viel mehr für mich getan hat, als sie ahnt. Außerdem bei Nathalie, Tone, Clara, Michi, Rachel, Jasmin, Sven, Alev und alle anderen, die mir in den letzten fünf Jahren zur Seite standen.

Danke!

Summary

Abstract

Traditional networking devices support only fixed features and limited configurability. Network softwarization leverages programmable software and hardware platforms to remove those limitations. In this context the concept of programmable data planes allows directly to program the packet processing pipeline of networking devices and create custom control plane algorithms. This flexibility enables the design of novel networking mechanisms where the status quo struggles to meet high demands of next-generation networks like 5G, Internet of Things, cloud computing, and industry 4.0. P4 is the most popular technology to implement programmable data planes.

However, programmable data planes, and in particular, the P4 technology, emerged only recently. Thus, P4 support for some well-established networking concepts is still lacking and several issues remain unsolved due to the different characteristics of programmable data planes in comparison to traditional networking. The research of this thesis focuses on two open issues of programmable data planes. First, it develops resilient and efficient forwarding mechanisms for the P4 data plane as there are no satisfying state of the art best practices yet. Second, it enables Bit Index Explicit Replication (BIER) in high-performance P4 data planes. BIER is a novel, scalable, and efficient transport mechanism for IP multicast traffic which has only very limited support of high-performance forwarding platforms yet.

The main results of this thesis are published as 8 peer-reviewed and one post-publication peer-reviewed publication. The results cover the development of suitable resilience mechanisms for P4 data planes, the development and implementation of resilient BIER forwarding in P4, and the extensive evaluations of all developed and implemented mechanisms. Furthermore, the results contain a comprehensive P4 literature study. Two more peer-reviewed papers contain additional content that is not directly related

to the main results. They implement congestion avoidance mechanisms in P4 and develop a scheduling concept to find cost-optimized load schedules based on day-ahead forecasts.

The majority of the research related to the main results have been funded by the Deutsche Forschungsgemeinschaft (DFG) in the context of the research project „Future Internet Routing (FIR)“ under grand ME2727/1-2.

Kurzfassung

Herkömmliche Netzwerkgeräte unterstützen nur einen festen Funktionsumfang und haben begrenzte Konfigurierbarkeit. Network softwarization nutzt programmierbare Soft ware- und Hardware-Plattformen, um diese Einschränkungen aufzuheben. In diesem Kontext erlaubt das Konzept der programmable data planes die direkte Programmierung der Paketverarbeitungspipeline von Netzwerkgeräten und die Verwendung benutzerdefinierter Algorithmen für die control plane. Diese Flexibilität ermöglicht die Entwicklung neuartiger Netzwerkmechanismen, wo der Status quo Schwierigkeiten hat, die hohen Anforderungen von Netzwerken der nächsten Generation wie 5G, Internet of Things, Cloud Computing und Industrie 4.0 zu erfüllen. P4 ist die am Weitesten verbreitete Technologie zur Implementierung von programmable data planes.

Programmable data planes und insbesondere die P4-Technologie wurden jedoch erst vor kurzem entwickelt. Daher fehlt es noch an P4-Unterstützung für einige etablierte Netzwerkkonzepte, und mehrere Probleme blieben bisher aufgrund der unterschiedlichen Eigenschaften von programmable data planes im Vergleich zu traditionellen Netzwerken ungelöst. Die Forschung in dieser Arbeit konzentriert sich auf zwei offene Fragen zu programmable data planes. Erstens werden widerstandsfähige und effiziente Weiterleitungsmechanismen für die P4 data plane entworfen und implementiert, da es bisher noch keine zufriedenstellenden Best Practices gibt. Zweitens wird BIER für hochleistungsfähigen P4 data planes entwickelt. Bit Index Explicit Replication (BIER) ist ein neuartiger, skalierbarer und effizienter Transportmechanismus für IP-Multicast-Verkehr, der bisher nur in sehr begrenztem Umfang von Hochleistungsgeräten unterstützt wird.

Die Hauptergebnisse dieser Arbeit wurden in 8 peer-review und einer post-publication-peer-review Publikation veröffentlicht. Die Ergebnisse umfassen die Entwicklung geeigneter Resilienz-Mechanismen für die P4 data planes, die Entwicklung und Imple-

mentierung von widerstandsfähiger BIER-Weiterleitung in P4 und die umfangreichen Evaluierungen aller entwickelten und implementierten Mechanismen. Darüber hinaus enthält die Forschungsarbeit eine umfassende P4-Literaturstudie. Zwei weitere peer-review Arbeiten enthalten zusätzliche Inhalte, die nicht direkt mit den Hauptergebnissen dieser Arbeit zusammenhängen. Sie implementieren Mechanismen zur Vermeidung von congestion in der data plane in P4 und entwickeln ein Planungskonzept, um kostenoptimierte Lastpläne auf Basis von Day-Ahead-Prognosen zu erstellen.

Der Großteil der Forschungsarbeiten zu den Hauptergebnissen wurde von der Deutschen Forschungsgemeinschaft (DFG) im Rahmen des Forschungsprojekts „Future Internet Routing (FIR)“ unter dem Förderkennzeichen ME2727/1-2 gefördert.

List of Publications

My personal contributions to all publications (§ 6 Abs. 2 Satz 3 der Promotionsordnung) can be found in the appendix.

Accepted Manuscripts (Core Content)

1. Daniel Merling, Steffen Lindner, and Michael Menth. **Robust LFA Protection for Software-Defined Networks (RoLPS)** [MLM21b]. *IEEE Transactions on Network and Service Management (TNSM)*, vol. 18, pp. 2570-2586, 2021. The published version of this publication can be found in the Appendix 1.1. The paper is also available online at the following URL: `https://doi.org/10.1109/TNSM.2021.3090843`

2. Steffen Lindner, Daniel Merling, Marco Häberle, and Michael Menth. **P4-Protect: 1+1 Path Protection for P4** [LMHM20]. *Proceedings of the 3rd P4 Workshop in Europe*, pp. 21-27, 2020. The published version of this publication can be found in the Appendix 1.2. The paper is also available online at the following URL: `https://doi.org/10.1145/3426744.3431327`

3. Daniel Merling, Michael Menth, Nils Warnke, Toerless Eckert. **An Overview of Bit Index Explicit Replication (BIER)** [MMWE18]. *IETF Journal*, 2018. The published version of this publication can be found in the Appendix 1.3 and online at the following URL: `https://www.ietfjournal.org/an-overview-of-bit-index-explicit-replication-bier/`. This publication has been subject only to post-publication peer review.

4. Daniel Merling, Steffen Lindner, and Michael Menth. **Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast** [MLM20a]. *Proceedings of Seventh International Conference on Software Defined Systems (SDS)*, Paris, France, 2020, pp. 51-58. The published version of this publication can be found

in the Appendix 1.4. The paper is also available online at the following URL: `https://doi.org/10.1109/SDS49854.2020.9143935`.

5. Daniel Merling, Steffen Lindner, and Michael Menth. **P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast** [MLM20b]. *Journal of Network and Computer Applications (JNCA)*, vol. 169, 2020. The published version of this publication can be found in the Appendix 1.5. The paper is also available online at the following URL: `https://doi.org/10.1016/j.jnca.2020.102764`

6. Daniel Merling, Steffen Lindner, and Michael Menth. **Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4** [MLM21a]. *IEEE Access*, vol. 9, pp. 34500-34514, 2021. The published version of this publication can be found in the Appendix 1.6. The paper is also available online at the following URL: `https://doi.org/10.1109/ACCESS.2021.3061763`

7. Daniel Merling, Thomas Stüber, and Michael Menth. **Efficiency of BIER Multicast in Large Networks** [MSM22]. *Accepted for publication in IEEE Transactions on Network and Service Management (TNSM)*. The most recent version of this publication can be found in the Appendix 1.7.

8. Steffen Lindner, Daniel Merling, and Michael Menth. **Learning Multicast Patterns for Efficient BIER Forwarding with P4** [LMM22]. *Accepted for publication in IEEE Transactions on Network and Service Management (TNSM)*. The most recent version of this publication can be found in the Appendix 1.8.

9. Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. **A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research** [HHM+23]. *Journal of Network and Computer Applications (JNCA)*, vol. 212, 2023. The most recent version of this publication can be found in the Appendix 1.9. It is also available as preprint on arXiv: `https://arxiv.org/abs/2101.10632`.

Accepted Manuscripts (Additional Content)

10. Michael Menth, Habib Mostafaei, Daniel Merling, and Marco Häberle. **Implementation and Evaluation of Activity-Based Congestion Management Using**

P4 (P4-ABC) [MMMH19]. *Future Internet*, vol. 11, 2019. The published version of this publication can be found in the Appendix 2.1. The paper is also available online at the following URL: `https://doi.org/10.3390/fi11070159`

11. Florian Heimgärtner, Sascha Heider, Thomas Stüber, Daniel Merling, and Michael Menth. **Load Profile Negotiation in Day-Ahead Planning for Compliance with Power Limits** [HHS$^+$19]. *Proceedings of the ETG Kongress*, pp. 1-6, 2019. The published version of this publication can be found in the Appendix 2.2. The paper is also available online at the following URL: `https://ieeexplore.ieee.org/document/8836012`

1 Introduction & Overview

In the past years, network softwarization became an important research field in the area of communication networks. It offers solutions where the status quo struggles to meet latency, bandwidth, flexibility, and availability demands of next-generation communication networks like 5G, Internet of things, cloud computing, and industry 4.0.

Traditional networks consist of devices, e.g., routers, switches, or firewalls, with support for only very specific protocols, and mechanisms. Other functionalities that may be desired by network operators are not supported and manufacturers allow only very limited configurability of the devices.

Network softwarization is a novel networking paradigm that reduces the dependence on the manufacturers. It enables new flexibility for network operators to design and implement networking services, protocols, and functions. The core idea is that network devices become programmable. That is, their functionality can be adapted by network operators without the need to change the underlying platform or buy other devices when demands change.

In the following, I will further specify the context of my thesis and describe its objectives. That is, I will introduce three highly-relevant concepts and technologies, i.e., software-defined networking (SDN), data plane programming with P4, and BIER. Afterwards, I describe the objectives of this thesis and briefly summarize its results.

1.1 Software-Defined Networking and Data Plane Programming

Software-defined networking (SDN) is a well-known and important concept in the context of network softwarization. Figure 1.1 compares SDN and traditional networking.

Networking devices consists of a packet processor, i.e., the data plane, which is steered by the control plane. In traditional networking devices data plane and control plane

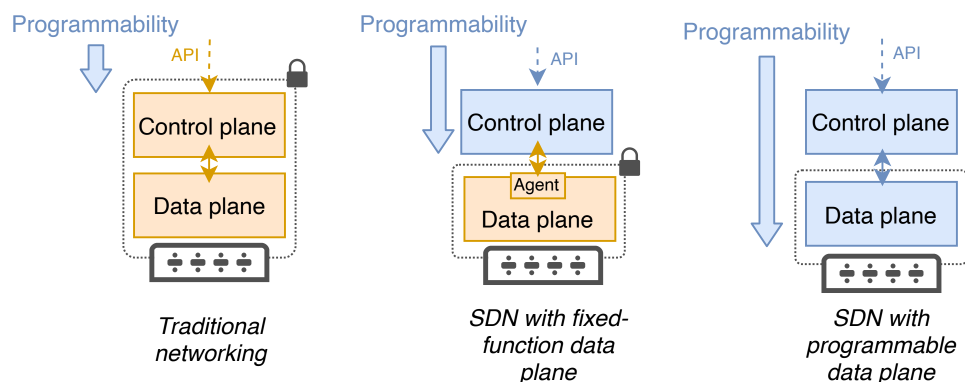


Figure 1.1: Comparison of traditional networking, SDN with fixed-function data planes, and SDN with programmable data planes (figure from Hauser et al. [HHM+23]).

are tightly coupled and often located on the same device. The control plane offers user interfaces like web interfaces, command line interfaces (CLIs), or application programming interfaces (APIs) through which network operators configure the device. However, configuration is limited to specific actions through predefined interfaces by the manufacturer and there is no direct access to either control plane or data plane.

SDN decouples data plane and control plane. That is, the data plane is extended by an API that allows network operators to directly configure it with custom software-based control plane algorithms. This significantly increases flexibility because the complex distributed control plane of traditional networking devices can be replaced by a centralized control plane with a comprehensive view of the network and custom functionalities. Network operators program the control plane so that it configures the data plane by calling predefined actions through the data plane API. This is called SDN with a fixed function data plane. OpenFlow (OF) [MAB+08] is the most popular SDN technology with fixed function data planes.

However, flexibility is still limited since API and actions are predefined and there is still no direct access to the data plane. The next step in introducing more flexibility is data plane programming, i.e., SDN with a programmable data plane. There, the data plane is described by software, e.g., a program, and custom APIs allow its configuration during network runtime by a custom control plane. P4 (Programming Protocol-Independent Packet Processors) [BDG+14] is the most widespread data plane programming technology.

P4 and data plane programming in general, have a disrupting impact on building and

operating communication networks. It enables full flexibility, i.e., data plane, control plane, and APIs are defined by the network operators. Programmable, high-performance hardware gives users the opportunity to create custom packet processing algorithms and control mechanisms without manufacturer-dependence. This enables many opportunities for innovation in industry and academia.

1.2 Programmable Protocol-Independent Packet Processor (P4)

P4 [BDG+14] is both an architecture and programming language to describe data planes. We published a comprehensive literature study on P4 [HHM21] with 519 references which is also part of this thesis. It includes thorough descriptions of the P4 ecosystem, tutorials, examples, and comparisons to other technologies so that readers can easily place P4 in the wider context of SDN. Furthermore, we reviewed hundreds of research papers with and on P4 and summarize their contents. The following part briefly summarizes the paper and gives an introduction to P4.

P4 is a high-level programming language to describe data planes. Target-specific compilers translate the P4 programs into binaries for the particular P4-programmable device, i.e., P4 target. To that end, P4 leverages an abstract model for the packet processing pipeline, which is shown in Figure 1.2.

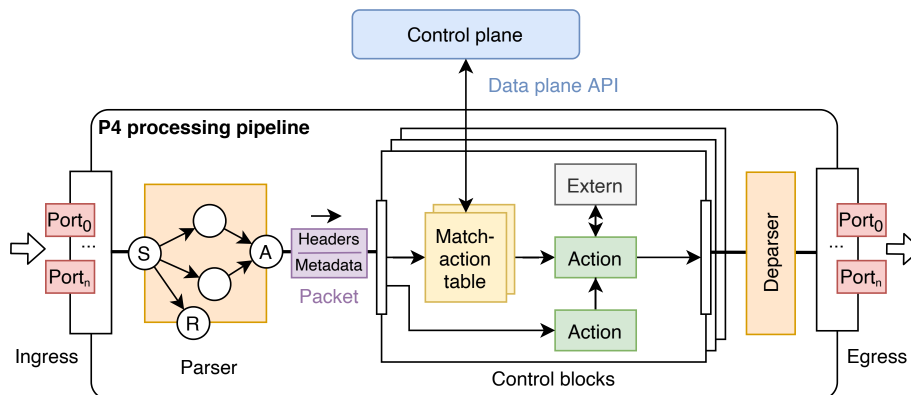


Figure 1.2: Simplified P4 processing pipeline (adjusted from Figure 5 from Hauser et al. [HHSM20]).

The P4 processing pipeline consists of a programmable parser, a match-action pipeline, and a programmable deparser. The parser is based on a programmable finite-state machine. Users can define custom headers and parsing schemes, e.g., for different header

structures. The parser deseralizes the packet and extracts header information, custom metadata, e.g., intermediate calculations, and intrinsic metadata, e.g., timestamps and ingress port number, into variable-like fields for later processing.

The match-action pipeline consists of programmable match-action-tables (MATs). They decide which actions are applied to a packet. That is, header or metadata fields are compared with entries of a table and upon a match a predefined action is executed. Users can define custom actions, e.g., dropping a packet or changing header fields, match fields, e.g., the destination IP address in the packet header, and parameters. The P4 program describes the structure of the MAT, i.e., the match fields and actions. The MAT is then filled with entries by the control plane. This gives the users full control over the behavior of the packet processing pipeline and even complex, novel packet forwarding and protocols can be implemented. P4 targets may introduce platform-specific actions and functionalities, e.g., registers, cryptographic functions, or port monitoring. Finally, the deparser serializes the packet according to the processing in the match-action pipeline.

1.3 Bit Index Explicit Replication (BIER)

IP multicast (IPMC) delivers one-to-many traffic like live-streaming, financial broadcast data, IPTV, or multicast VPN. To that end, IPMC is organized into multicast groups. Core devices maintain multicast group-specific information to which neighbors packets should be forwarded. Thus, traditional IPMC has three scalability issues. First, forwarding information occupy extensive storage of core devices. Second, whenever IPMC groups change, forwarding information is updated, which requires significant signalling effort. Third, when the topology changes or a link or node fails, many devices may require an update.

Bit Index Explicit Replication (BIER) [WRD+17] was proposed by the Internet Engineering Task Force (IETF) as an efficient and scalable transport mechanism for multicast traffic that does not suffer the aforementioned downsides. That is, BIER moves forwarding information for IPMC groups from core devices to the packet header to keep the core network stateless. Section 2.2.1 contains a detailed introduction of BIER.

1.4 Research Objectives

SDN, and in particular, programmable data planes, have significant advantages over traditional networks. However, both technologies are relatively new. Thus, several issues remain unsolved, which reduce efficiency and opportunities. This thesis focuses on two main objectives that tackle open issues of SDN and programmable data planes.

The first objective is to develop **resilient forwarding mechanisms for the data plane in SDN**. Legacy networks have a variety of different resilience mechanisms to harden packet forwarding against failures. Due to the different networking paradigm in SDN, some mechanisms cannot be easily transferred from legacy to SDN, or their operation is inefficient. As a result, there are no satisfying state of the art best practices for resilience of data plane traffic in SDN.

This objective is separated into three main issues. First, we design suitable mechanisms with the characteristics of SDN in mind, e.g., separated data plane and control plane. Second, those mechanisms should be implementable in P4, the state of the art programmable data plane technology. This is a serious challenge due to the limited hardware support and strong constraints of existing platforms on packet forwarding operations to maintain high forwarding rates. Finally, we perform comprehensive evaluations of the designed mechanisms to show their capabilities and limitations.

The second objective of this thesis is the **support of BIER in P4**. SDN platforms and P4 support traditional multicast with its limitations described in Section 1.3. At the time of writing this thesis, BIER is not natively supported by any SDN platform. However, the interest in BIER by industry and academia is significantly increasing and serious undertakings by the IETF BIER working group aim at developing BIER even further to make it the state of the art multicast transport mechanism. The goals of this objective are twofold.

First, we develop a BIER-based forwarding scheme in P4. This enables BIER-based forwarding and its advantages in comparison to traditional IPMC in SDN. Second, we perform extensive evaluations of BIER in both, simulative environments, and hardware-based testbeds, with regard to its efficiency and scalability.

1.5 Research Context

Most research in this thesis was funded by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. This research project with the title „Future Internet Routing (FIR)“ aims at developing and evaluating novel forwarding mechanisms with respect to demands of novel network infrastructures.

All research was done in collaboration with colleagues. A description of the contributions to the overall work by my colleagues and me can be found in the appendix.

1.6 Research Results

This thesis summarizes the work of 11 publications which can be found in the appendix. Chapter 2 describes and discusses research with respect to the two main objectives and additional secondary results of this thesis. Each main objective contains multiple contributions. We explain the context and foundations, describe the research approach and the results, and discuss their relevance. Now we give an overview on the research and the results.

One contribution of this thesis is the P4 literature study [HHM+23]. It contains a comprehensive overview of research on P4 and projects with P4. Especially from the review of existing resilience and multicast mechanisms we derived the open issues of SDN with programmable data planes, and therefore, the two main objectives of this thesis. Thus, the P4 survey summarizes related work and analyzes the two aforementioned research areas to gain meaningful insights and identify open research potential. Throughout the remainder of this document, it serves as a reference to place the research in a wider context and is not describe any further.

The first main objective, i.e., **resilience in SDN with P4**, is covered in Section 2.1. One significant contribution is the development of fast 1:1 protection in SDN based on P4 with extensive evaluations [MLM21b] in Section 2.1.1. We developed a local rerouting scheme, based on popular legacy technology, that protects against any single component failure and most double failures, and detects and stops looping traffic, i.e., loops, in any failure scenario. Thereby, it requires only very little overhead, i.e., at most only one additional tunnel header, and only very few additional forwarding entries in core nodes in the network. The evaluations show that it outperforms state of the art legacy technologies and other SDN-based resilience mechanisms. Furthermore,

we implemented this mechanism on high-performance P4-programmable hardware and showed its feasibility in a hardware-based 100 Gb/s line rate[1] testbed. The results show that downtime due to failures can be reduces to under 1 ms, and that loops are detected and prevented correctly.

The second contribution is the development of a 1+1 protection scheme in P4 [LMHM20] in Section 2.1.2. Thereby, traffic is forwarded over two disjoint paths so that traffic is delivered even if one path is interrupted. 1+1 protection is preferably used in scenarios with time-sensitive traffic, like live streaming. The evaluations prove its feasibility in a hardware-based testbed.

The source codes of both projects is publicly available for the benefit of fellow researches and developers. Furthermore, results were discussed with researchers from industry, e.g., on IETF, and/or joint projects.

The second main objective, i.e., **BIER support in P4**, is described in Section 2.2. The first contribution is the successful development of BIER prototypes in P4 [MLM20b, MLM21a]. Due to the complex nature of BIER forwarding this proved to be a serious challenge. We describe the process and the results in detail in Section 2.2.4. We implemented BIER and BIER fast reroute (BIER-FRR) [MLM20a], a novel BIER resilience scheme that is a contribution of this thesis (see Section 2.2.2), for a P4-programmable software switch, and a high-performance, P4-programmable switching ASIC with 100 Gb/s line rate. The results show the feasibility of BIER on 100 Gb/s line rate P4 hardware with the overhead of capacity issues (see Section 2.2.4.2.1). That is, when BIER traffic arrives at high rates and has many next-hops, additional processing capacity may be required to prevent packet loss. However, the evaluations show that for realistic traffic mixes of unicast and multicast, only very little additional capacity is required (see Section 2.2.4.2.2).

A further contribution is the evaluation of the efficiency of BIER and BIER-FRR and their scalability (see Section 2.2.3) [MSM22]. We showed that IPMC reduces the one-to-many traffic amount in comparison to unicast. BIER is less efficient than IPMC, but generates still significantly less traffic amount than unicast. We conducted comprehensive evaluations to show the efficiency depends on different topologies and configuration properties.

Finally, we enhanced BIER-based forwarding in P4 so that even less capacity is required to prevent packet loss (see Section 2.2.4.3) [LMM22]. To that end, we lever-

[1]Line rate is the typical communication speed of two connected devices.

age machine learning strategies to learn traffic patterns so that forwarding can be optimized.

All implementations are open-source, and the results were frequently presented and discussed in the IETF and at other occasions.

Finally, Section 2.3 contains additional research without direct relation to the aforementioned main objectives of this thesis. In Section 2.3.1 we implemented activity-based congestion management (ABC) for a P4 software platform and performed evaluations [MMMH19]. We showed that ABC successfully achieves fairness among users for both UDP and TCP traffic, even when some users try to actively gain an unfair advantage over others. Thereby, ABC does not require per-flow or per-user state in core nodes. In Section 2.3.2 we developed models to leverage flexibility in energy demands and price forecasts to build operation schedulings that reduce or even minimize the overall energy cost [HHS+19].

2 Results & Discussion

This chapter presents and discusses the results of this thesis. Section 2.1 contains research on the first main objective of this thesis, i.e., fast protection of data plane traffic in SDN with P4. Section 2.2 presents the second main objective which is research on BIER with the goal to implement it in P4. Section 2.3 covers additional research without direct relation to the main objectives of this thesis.

2.1 Protection of Data Plane Traffic in SDN with P4

Failure of network components have a negative impact on performance, e.g., availability of services, throughput, or latency. Default forwarding paths are interrupted and data packets may be dropped when the next-hop (NH) is unreachable. However, recomputation of paths and updating the networking devices requires a considerable amount of time.

To avoid packet loss in the meantime, legacy networks often leverage fast reroute (FRR) mechanisms to reroute packets on precomputed backup paths during recomputation. Raj et al. [RI07], Rai et al. [RMD05], and Papan et al. [PSPM17] present surveys with a wide overview on FRR mechanisms for legacy networks. Santos da Silva et al. [dSMSF15], and Chiesa et al. [CKR+21] survey FRR mechanisms for SDN. For a detailed discussion on the content of those surveys, see the related work section of Merling et al. [MLM21b]. We summarize that most mentioned mechanisms are either complex, restricted to specific forwarding technologies or use-cases, require extensive overhead, protect only the control plane, or are not easily applicable to SDN.

In the following, we summarize and discuss the research results of this thesis for efficient data plane protection in SDN. First, we present Loop-Free Alternate (LFA)-based 1:1 protection in P4 that locally reroutes traffic around the failure. Afterwards, we describe 1+1 protection in P4 where traffic is always sent on two disjoint paths to the destination. Finally, we discuss future research.

2.1.1 Robust LFA Protection for Software-Defined Networks (RoLPS)

In this section, we summarize and present the most important research results from Merling et al. [MLM21b]. For more details please refer to the paper. First, we introduce important foundations and the problem statement. Then, we present the research objectives, the concept and implementation of Robust LFA Protection for Software-Defined Networks (RoLPS), and evaluation results.

2.1.1.1 Foundations

We introduce LFAs, and describe loop detection.

2.1.1.1.1 Loop-Free Alternates Loop-Free Alternates (LFAs) [AZ08] are a well-known FRR mechanism in legacy networks and are often used due to their simplicity and low overhead. The point of local repair (PLR) locally reroutes packets to an alternative NH if the default NH is unreachable. The alternative NH is called LFA and it has to be selected in a way that it still has a working shortest path towards the destination despite the failure.

LFAs can be link protecting or node protecting, while the latter also includes the former. An LFA with link protection (LP) protects against the failure of the link between PLR and NH by avoiding that link on its path towards the destination. An LFA with node protection (NP) avoids the failed NH. However, LFAs cannot protect all destinations.

Remote LFAs (rLFAs) [BFP+15, SHB+17, CR15] increase the number of protected destinations. rLFAs are remote nodes in the network that can still reach the destination. The PLR adds a tunnel header to the affected packet such that it is rerouted over shortest paths to the rLFA instead. There, the tunnel header is removed and the original packet is forwarded towards the destination. There are LP rLFAs and NP rLFAs.

Finally, there are topology-independent LFAs (TI-LFAs) [LBF+22, FB17] with LP or NP. TI-LFAs leverage an address stack of multiple intermediate nodes to explicitly define paths towards remote nodes in the network that have a working shortest path towards the destination.

2.1.1.1.2 Loop Detection Braun et al. [BM16] found that all LFAs may cause loops under severe failure conditions. Loops should be heavily avoided because they may easily occupy 30-fold of the capacity of regular traffic. Therefore, Braun et al. [BM16] present a loop detection mechanism based on a bitstring. Each node is assigned to a position in that bitstring. Whenever a node reroutes a packet, it activates its bit. When a node has to reroute a packet, but its own bit is already activated, the packet is dropped instead. This mechanism requires significant header space in large networks.

2.1.1.2 Research Objectives

The objective was to create efficient and robust best practices for protection of data plane traffic in SDN. We selected LFAs as a basis due to their popularity in legacy networks, and their low overhead compared to other approaches. This objective consists of three goals.

The first goal was to design an LFA-based mechanism that protects all destinations against both single link failures (SLFs) and single node failures (SNFs) with only low overhead. The research results in this thesis [MLM21b] and Braun et al. [BM16] show that both LFAs and rLFAs cannot provide sufficient protection, i.e., protect all destinations against both SLFs and SNFs all networks. We discuss those results in detail in Section 2.1.1.4.2. TI-LFAs protect all destinations, but we showed[1] that they may be inefficient due to large header stacks [MLM21b]. Therefore, we did not consider TI-LFAs any further. As a results, we developed a new type of LFAs with the desired protection capabilities. We call them explicit LFAs (eLFAs) and present them in Section 2.1.1.3.1.

The second goal was to design a suitable loop detection mechanism for SDN. In Section 2.1.1.1.2 we presented the solution of Braun et al. [BM16]. However, this approach may cause large headers because each node in the network requires a bit position. The authors argue that bit positions may be reused to prevent large headers but this may cause false positive detection of loops when different nodes that share a bit position reroute a packet. Thus, we designed a more efficient loop-detection and prevention mechanism which we present in Section 2.1.1.3.2.

The third goal was to implement this protection scheme in P4 and evaluate its efficiency and feasibility. The target platform was a high-performance P4-programmable

[1] See Section V-D of Merling et al. [MLM21b])

switching ASIC with up to 3.2 Tb/s switching capacity, i.e., 100 Gb/s per link. This is a notable contribution because most other P4-based FRR implementations which we reviewed extensively in Hauser et al. [HHM+23], target only low-performance platforms as high-performance targets impose additional constrictions on packet processing to guarantee high throughput. We present the implementation concept in Section 2.1.1.5.

2.1.1.3 Robust LFA Protection for Software-Defined Networks (RoLPS)

In this section, we present Robust LFA Protection for Software-Defined Networks (RoLPS) [MLM21b]. First, we introduce the novel explicit LFAs (eLFAs) and advanced loop detection (ALD). Then, we explain the concept of RoLPS-based protection.

2.1.1.3.1 Explicit LFAs Explicit LFAs (eLFAs) are remote nodes in the network. The difference to rLFAs is that eLFAs are not restricted to shortest paths. Instead, the PLR reaches an eLFA through an explicit tunnel. Thus, eLFAs can be nodes that are not reachable via shortest paths anymore. Figure 2.1 shows an example topology with link costs and a link failure where the PLR sends a packet towards the destination D. First, we explain why LFAs and rLFAs cannot protect this destination. N is not an

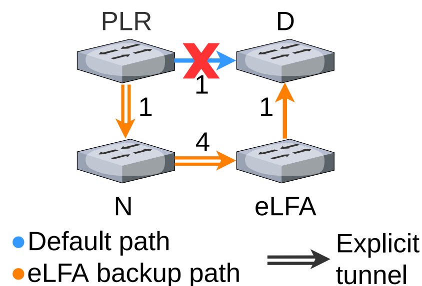


Figure 2.1: eLFAs leverage an explicit tunnel to send the packet to an arbitrary node regardless of link costs (figure from Merling et al. [MLM21b]).

LFA because it forwards traffic towards D back to the PLR which creates a traffic loop between PLR and N. Furthermore, the node eLFA is not an rLFA because the shortest path from the PLR to the eLFA crosses the broken link. Therefore, we leverage an explicit tunnel from the PLR over N to the eLFA.

Explicit tunnels are implemented with additional forwarding entries in tunnel nodes. To that end, the eLFA gets an unique IP address. When the PLR reroutes a packet, it

adds a tunnel header to the packet. The IP address in the tunnel header is the unique IP of the eLFA. Nodes on the tunnel path forward the packet according to tunnel-specific forwarding entries. The eLFA removes the tunnel header and forwards the original packet

2.1.1.3.2 Advanced Loop Detection The loop detection from Braun et al. [BM16] requires large headers or may drop packets erroneously. We present advanced loop detection (ALD) which requires significantly less header space.

Advanced loop detection (ALD) requires a counter in the packet header. It allows packets to be rerouted n times. Upon the next reroute, the packet is dropped instead. Users may configure ALD with a large n, but we perform the evaluations in this thesis with $n = 2$ which is implementable by a single bit. We show in Section 2.1.1.4.2 that this is sufficient to detect and drop all loops in single failure scenarios and most loops in scenarios with multiple failures.

2.1.1.3.3 Concept of RoLPS-Based Protection RoLPS leverages the novel eLFAs and ALD to protect data plane traffic against any single component failure. To reduce overhead, it relies on less complex LFA-types if possible. For example, an eLFA is used only when there is no LFA or rLFA to protect a destination. This prevents unnecessary additional forwarding entries to implement explicit tunnels. ALD is used to detect and stop loops under any failure condition. RoLPS can be configured with different protection degrees and complexity levels. For details see Merling et al. [MLM21b]. In the following we present results only for RoLPS with LP and RoLPS with NP which guarantee the highest degrees of protection.

2.1.1.4 Simulative Performance Evaluation

First, we describe the methodology. Then, we evaluate protection coverage and overhead in terms of additional forwarding entries.

2.1.1.4.1 Methodology We evaluate several RoLPS-based protection on 205 commercial, wide area, academic, and research networks from the Internet topology zoo [KNF$^+$11] and three data center topologies (DCell, fat-tree, BCube). We include link costs in the evaluations because those networks are considered more challenging than

networks without link costs[2]. To that end, we leverage a full traffic matrix based on shortest path. That is, each node sends a packet to each other node on shortest paths and we count the number of packets that travel over each link, i.e., the link load. The link cost of a link is the inverse of the link load multiplied by the larges link load in the network[3]. For each topology we forward packets according to shortest paths[4]

2.1.1.4.2 Protection Coverage We consider four failure scenarios to evaluate LFAs, rLFAs, and RoLPS. A failure scenario $\mathcal{S} \in \{\mathrm{SLF}, \mathrm{SNF}, \mathrm{DLF}, \mathrm{SLF{+}SNF}\}$ consists of either all SLFs, SNFs, double link failures (DLFs), or single link failures plus single node failures (SLF+SNF). We evaluate a full traffic matrix, i.e., a flow from each source to each destination, and apply every failure from $\mathcal{S}$ that affects that flow. A flow may be protected, unprotected or loop when it is affected by a failure. A flow is „protected“ when the packet is successfully delivered due to rerouting. A packet may be dropped for two reasons. First, if the packet is dropped and there is no physical path to the destination we consider this scenario also as „protected“ because the packet would loop otherwise. Second, if the packet is dropped although the destination is still reachable, this scenario is considered „unprotected“. Finally, a packet may „loop“. The „coverage“ is the fraction of protected destinations. We summarize the results for every considered FRR mechanism over all topologies and report the fraction of protected, unprotected, and looped flows in a bar chart.

Figure 2.2 shows the results on the 208 topologies with link costs. We see the benefit of rLFAs[5] over LFAs[5] in SLFs scenarios, i.e., 87% over 60% protected destinations. However, only RoLPS-based mechanisms are able to protect all destinations. In SNFs scenarios both, LFAs and rLFAs create a significant amount of loops, and both cannot protect all destinations. RoLPS with LP protects only 93% of destinations while RoLPS with NP achieves 100%. Both RoLPS mechanisms prevent loops.

In addition, we investigate protection coverage in scenarios with multiple failures although such scenarios are often not considered in FRR because traffic is rerouted only locally without further communication with other nodes. We see, that for DLFs, both

[2] Merling et al. [MLM21b] contains evaluations on the same topologies without link costs. The results support the claims we present here.

[3] This normalizes link costs so that the smallest link cost is 1. More details on link cost computation are found in Section V.A.3) of Merling et al. [MLM21b].

[4] RoLPS works for general destination-based forwarding but to reduce parameter space we limit evaluations to shortest paths.

[5] We consider only LFAs and rLFAs with LP because coverage with NP is even lower.

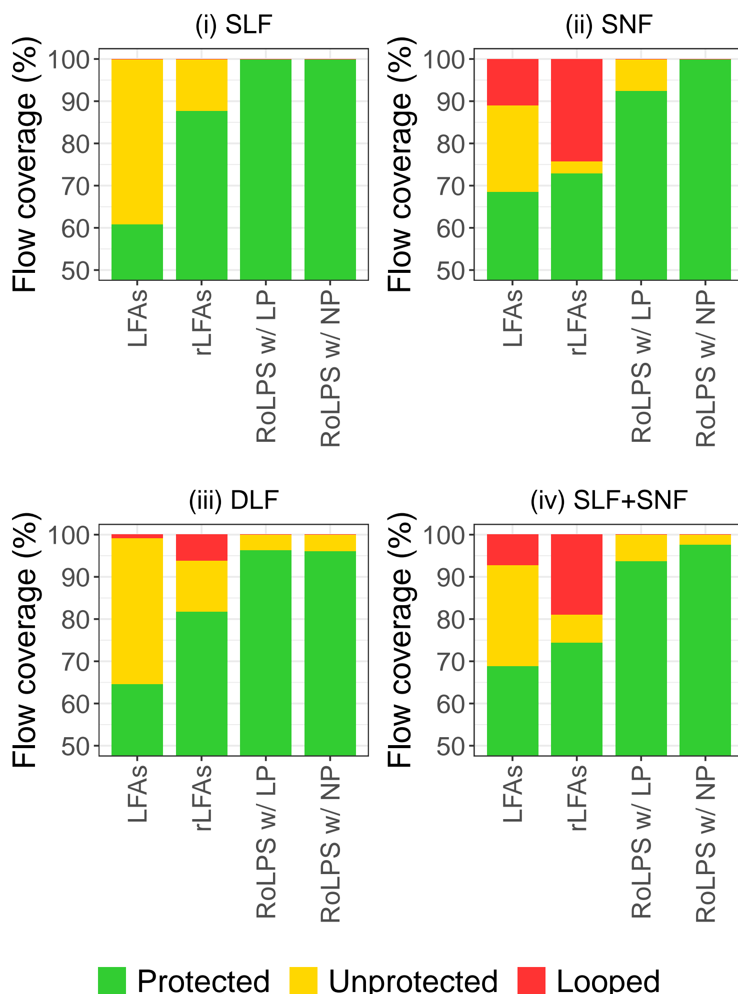


Figure 2.2: Fraction of protected, unprotected, and looped traffic over 208 topologies with link costs (adjusted from Figure 3.b) from Merling et al. [MLM21b]).

RoLPS with LP and RoLPS with NP, still protect around 95% of destinations while preventing loops. LFAs and rLFAs cause loops and protection a significant lower fraction of destinations, i.e., 65% and 82%. In the SLF+SNF scenario the results are similar.

2.1.1.4.3 Additional Forwarding Entries Each node maintains $n-1$ forwarding entries in a network with n nodes for shortest path forwarding. LFAs and rLFAs work with shortest paths and do not leverage additional forwarding entries which leads to lower coverage. Therefore, we evaluate the average and maximum amount of additional forwarding entries per node relative to $n-1$ for protection variants that require additional forwarding entries but protect all destinations. That is, RoLPS with LP and RoLPS with NP and Multiprotocol Label Switching (MPLS) with facility-backup. MPLS-facility-

backup (MPLS-FB) is a state of the art FRR mechanism which can be used in SDN. It has either LP, i.e., MPLS-FB-LP, or NP, i.e., MPLS-FB-NP, properties.

We evaluate the 208 topologies from the topology zoo with link costs[6]. We present the results as a complementary cumulative distribution function (CCDF). Figure 2.3 shows the results. RoLPS with LP requires only very few addtional forwarding entries. Only

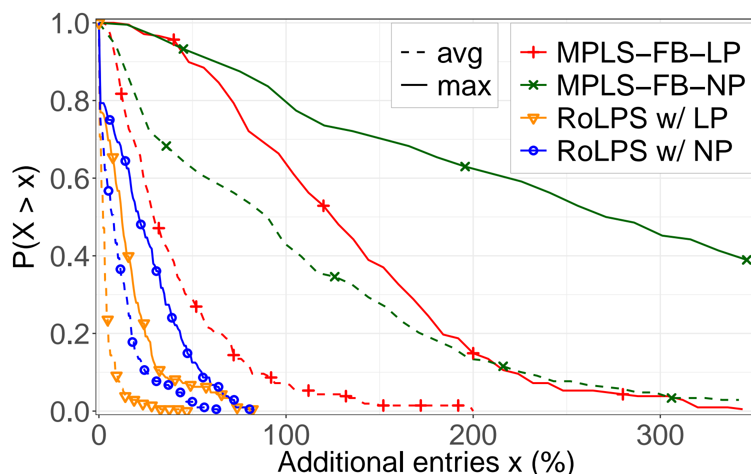


Figure 2.3: Relative increase in forwarding entries per node on 208 topologies with link costs (adjusted from Figure 4.b) from Merling et al. [MLM21b]).

10% of networks require more than 8% of additional entries on average. Furthermore, no network has a node that requires more than 80% of additional entries. MPLS-FB-LP causes in 20% of networks around 65% additional entries on average, and 40% of networks even have at least one node with 140% more additional entries.

RoLPS with NP installs only few additional entries. On average, 90% of nodes require less than 25% of additional entries. Furthermore, there in no network that has a node that requires more than 80% of additional state. MPLS-FB-NP causes at least 175% more additional entries on average in 20% of networks. 50% of networks even contain at least one node where number of entries increases by 300%.

2.1.1.5 RoLPS Implementation

In Hauser et al. [HHM21] we showed that most P4 projects are implemented for the P4 programmable software switch BMv2 [p4l19]. Most authors chose it as a target because

[6] see Merling et al. [MLM21b] for evaluations on topologies without link costs

it is extendable with custom packet processing actions when the actions from the P4 specification are insufficient to implement very complex packet processing behavior, like cryptographic operations or complex mathematical computations. However, the throughput of the BMv2 is limited to around 900 Mb/s [Bas18]. Furthermore, such modified P4 programs often cannot run on hardware targets with higher throughput because those cannot be extended in the same way.

Our target was the Intel Tofino [Int22] which is a P4-programmable high-performance switching ASIC with a capacity of 3.2 Tb/s. Such hardware targets impose additional restrictions on the P4 program, e.g., the number of operations per packet, to achieve high processing rates. Therefore, implementing a complex data plane for the Tofino is a significant challenge. The Tofino is used in the Edgecore Wedge 100BF-32X [Edg17] switch with 32 100 Gb/s ports which we use for our hardware-based evaluations.

The source code for the RoLPS data plane and control plane is publicly available[7]. Section VI of Merling et al. [MLM21b] contains more details about the implementation.

2.1.1.6 Hardware-Based Performance Evaluations

In this section, we present results of the hardware-based evaluation. First, we explain the evaluation setup for measurements of restoration time, followed by the results. Then, we evaluate the loop detection.

2.1.1.6.1 Setup for Measurements of Restoration Time Figure 2.4 shows the topology for the measurements of the restoration time. The evaluations focus on the Tofino (see Section 2.1.1.5). It is connected to a EXFO FTB-1 Pro traffic generator [EXF19] which generates up to 100 Gb/s of traffic[8]. Since we have access only to one Tofino we implement the rest of the network with BMv2s (see Section 2.1.1.5). Therefore, the traffic generator sends with only 100 Mb/s to not overload the BMv2s. This has no impact on restoration time measurements. BMv2-1 and BMv2-2 are two NHs of the Tofino. To resemble more realistic networks, we add five BMv2s and 10 links in the additional network.

All devices, except for the traffic generator, are connected to a controller. It sets up primary forwarding rules in all devices. We evaluate two scenarios. In the first scenario,

[7] `https://github.com/uni-tue-kn/p4-lfa`

[8] In a separate evaluation we confirmed that the RoLPS implementation achieves a throughput of 100 Gb/s on the Tofino with and without FRR. For details see Section VII.A. of Merling et al. [MLM21b].

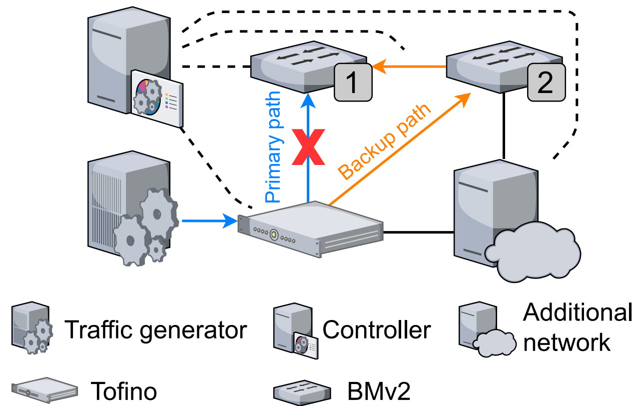


Figure 2.4: Topology for measurements of restoration time. The additional network consists of five other BMv2s and 10 links (figure from Merling et al. [MLM21b]).

the controller does not install any FRR rules. That is, when a device cannot forward a packet because the NH is unreachable, it notifies the controller which recomputes forwarding entries and updates the networking devices. In the second scenario, the controller installs RoLPS with LP so that the shown backup path is used by the Tofino when the primary NH is unreachable.

The traffic generator sends traffic to the Tofino which forwards the packets on the primary path to BMv2-1. Then, we disable the link between Tofino and BMv2-1 so that the primary path is interrupted. We measure the time until BMv2-2 receives traffic again after the deactivation of the primary link.

2.1.1.6.2 Restoration Time Measurements Figure 2.5 shows the average results of 10 runs and 95% confidence intervals. We see that without FRR the restoration time takes

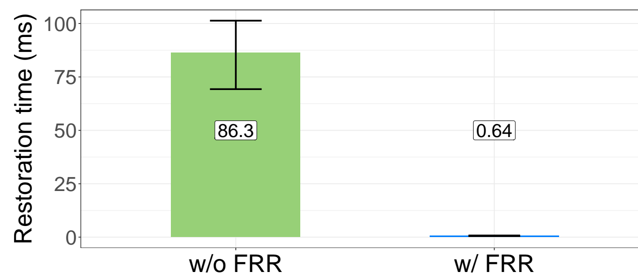


Figure 2.5: Restoration time with and without FRR (figure from Merling et al. [MLM21b]).

around 86.3 ms with around 15 ms fluctuations up or down. Note that we measured this time in a very small network with only 8 devices and a controller that does not perform any other task. In realistic networks with many devices and heavier load on the controller recomputation requires significantly more time possibly in the magnitude of multiple seconds.

When FRR is activated, the restoration time drops to under 1 ms because the Tofino immediately leverages backup forwarding rules to reroute the packet to BMv2-2 which then forwards the packet to BMv2-1.

2.1.1.6.3 Setup for Loop Detection Evaluation Figure 2.6 shows the testbed for evaluation of loop detection. The Tofino is connected to two BMv2s and a traffic generator.

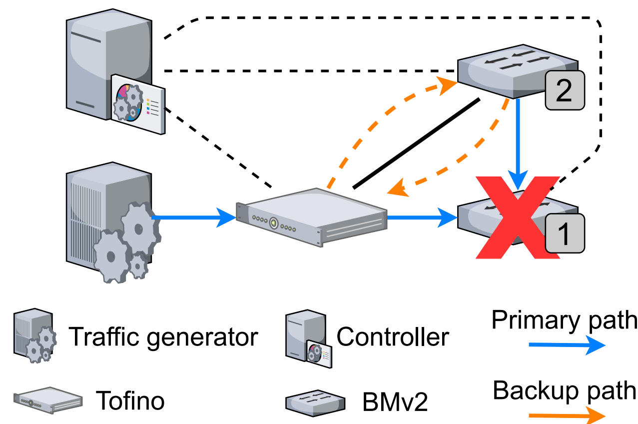


Figure 2.6: Setup for evaluation of loop detection (figure from Merling et al. [MLM21b]).

The traffic generator sends with only 100 Mb/s to not overload the BMv2s[9]. All devices, except the traffic generator, are connected to the controller. It sets up primary forwarding entries and RoLPS with LP in all devices.

We deactivate the primary NH, i.e., BMv2-1, to simulate a SNF. However, we installed RoLPS only with LP in the devices. Therefore, the Tofino will leverage its backup entries to forward the traffic to BMv2-2 upon failure detection. BMv2-2 will also use backup entries to send traffic back to the Tofino because the primary NH, i.e., BMv2-1, is unreachable. As a result, the packets will loop between the Tofino and BMv2-2.

[9]This has no impact on the evaluation of loop detection.

We evaluate two scenarios. In the first scenario, ALD is activated and allows one reroute (see Section 2.1.1.3.2). In the second scenario, ALD is deactivated. We report packet-ins at the BMv2-2.

2.1.1.6.4 Loop Detection Figure 2.7 shows the packet arrivals at BMv2-2.

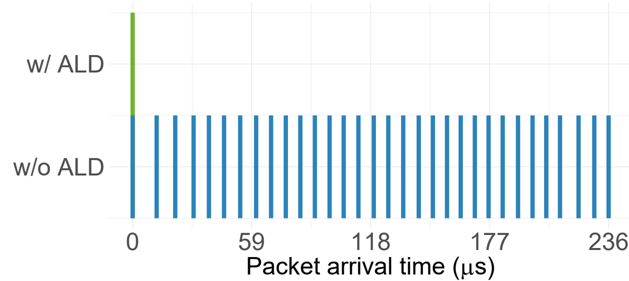


Figure 2.7: ALD successfully detects and drops loops (figure from Merling et al. [MLM21b]).

We see that ALD successfully detects and drops the loops. That is, BMv2-2 receives the packet after the first reroute from the Tofino. Then, it has to reroute the packet again because the default NH is unreachable. This exceeds the configured reroute limit and the packet is dropped. Without ALD, the packet loops between Tofino and BMv2-2 until its time-to-live (TTL) expires.

2.1.1.7 Conclusion and Discussion

We summarize the results of Merling et al. [MLM21b] and discuss future research.

2.1.1.7.1 Summary of the Results In Merling et al. [MLM21b] we presented a quick and efficient FRR mechanism for the SDN data plane. It requires only very low overhead in terms of additional forwarding entries compared to other state of the art mechanisms[10] and protects against all single component failures. Furthermore, it detects and stops all loops. We proved that RoLPS is implementable on state of the art high-performance P4-programmable hardware, i.e., the Tofino, and it achieves a throughput of 100 Gb/s per link. Furthermore, downtime after the detection of a failure is under 1 ms.

[10]In this summary we compared RoLPS only to MPLS-FB. Merling et al. [MLM21b] contains comparisons with more state of the art FRR mechanisms which support the results of this summary.

2.1.1.7.2 Outlook 1:1 protection like RoLPS requires quick detection of a failure. The Tofino offers a feature by which it detects port ups/downs to reroute packets quickly (for details, see Section VI.C.1) of Merling et al. [MLM21b]). However, this is a target-specific extension of the P4 features. That is, other targets require their own solutions. In particular, there may be targets that do not support fast detection of port up/down events. In such cases, operators need to establish bidirectional forwarding detections (BFDs) between nodes. Two connected nodes regularly exchange BFD packets to notify other participants about their liveliness. Future research could investigate on restoration times in realistic hardware-based networks with BFDs or other mechanisms when there is no target support for fast failure detection.

In general, RoLPS-based protection imposes only very little overhead in the network. However, the evaluations show that there may be topologies with high overhead. Future studies could evaluate the impact of topology structure, and other properties on the efficiency of RoLPS which were not part of this study.

Visibility of the Results

An early prototype of RoLPS has been developed during my masterthesis [Mer17] and published on the IEEE International Conference on Network Softwarization (IEEE NetSoft) [MBM18]. During my Ph.D. thesis, RoLPS has been implemented and significantly refined, extended, and evaluated. The concept, evaluations, and implementation were published as a journal paper in the Special Section on Design and Management of Reliable Communication Networks of IEEE Transactions on Network and Service Management [MLM21b] (Appendix 1.1) in 2021.

We presented and discussed the results on the KuVS Fachgespräch „Network Softwarization“ in April 2022, and the IETF MPLS working group.

The source code of the Tofino-based prototype with a throughput of up to 100 Gb/s per link and a restoration time of under 1 ms is publicly available[11].

2.1.2 P4 Protect

In this section, we present results from Lindner et al. [LMHM20]. I only briefly summarize the results because I was only secondary contributor to this work (see Sec-

[11] https://github.com/uni-tue-kn/p4-lfa

tion 1.6). First, I introduce the research objectives and then explain the solution concept.

2.1.2.1 Research Objectives

1:1 FRR mechanisms like RoLPS switch to a backup path at the PLR only when the default path is interrupted. Failure detection and rerouting may require some time. In contrast, with 1+1 protection traffic is always carried over two disjoint paths. When one path is interrupted, the destination still receives traffic over the unaffected path. This is advantageous for time-critical traffic, e.g., live-streaming or VoIP. However, 1+1 protection requires backup capacity even in the failure free case. Thus, both, 1:1 protection and 1+1 protection, are beneficial for different use cases.

With RoLPS, we already presented a suitable 1:1 protection scheme for programmable data planes. Therefore, the goal was to also develop a 1+1 protection mechanism for SDN with programmable data planes.

2.1.2.2 1+1 Protection Mechanism

P4-Protect is a P4-based 1+1 protection scheme [LMHM20]. Figure 2.8 shows its concept. A controller may set up a 1+1 protected connection between two endpoints.

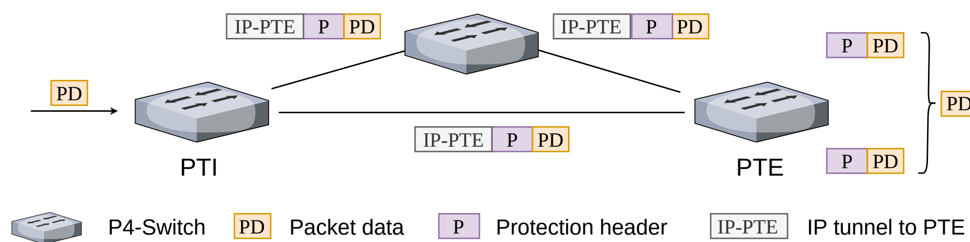


Figure 2.8: Concept of P4 protect (adjusted from Figure 1 from Lindner et al. [LMHM20]).

We call this the protection tunnel. The protection tunnel ingress (PTI) adds a protection header to the data packet and an IP header with the IP address of the protection tunnel egress (PTE). The protection header contains a sequence number so that the PTE can correctly order received packets. The packet is copied at the PTI and send over two disjoint paths towards the PTE. Path selection and implementation is discussed in more detail in Lindner et al. [LMHM20]. The PTE receives traffic on two paths. It leverages

the sequence number in the packet header to only accept in-order packets. Then, it discards packet duplicates, removes the protection header from the other packet and forwards the original payload further. If one path is interrupted by a failure, the PTE will still receive traffic over the other path which ensures very low downtimes which is crucial for time-sensitive traffic.

We implemented P4-Protect for the BMv2[12] and the Tofino[13]. The code is publicly available. The hardware-based evaluations with 100 Gb/s line rate show the feasibility of P4-Protect and its low overhead. For details, see Section 6 of Lindner et al. (2020) [LMHM20].

2.1.2.3 Future Research

P4-Protect is designed in a way that it even provides 1+1 protection over the Internet However, network operators have only very little influence on path selection and, in particular, disjointness. Real-world evaluations could accurately assess the efficiency of P4-Protect for such scenarios because isolated testbeds may omit several unknown conditions and restrictions of real infrastructure.

Visibility of the Results

We published P4-Protect [LMHM20] and comprehensive evaluations on the P4 Workshop in Europe (EuroP4) in 2020 (see Appendix 1.2). We presented it at the workshop and discussed the results. Furthermore, we made the implementation publicly available.

2.2 BIER-Based Multicast in P4

In this section, we summarize the research results of this thesis on BIER. More details, explanations, and evaluations can be found in the respective papers. First, we give an overview on BIER. Then, we present BIER fast reroute and evaluate the scalability and efficiency of BIER. Afterwards, we briefly describe the BIER implementation in P4 and present hardware-based performance evaluations. Finally, we discuss future research.

[12] https://github.com/uni-tue-kn/p4-protect
[13] https://github.com/uni-tue-kn/p4-protect-tofino

2.2.1 BIER Overview

In Section 1.3 we explained the downsides of state-of-the-art IPMC. That is, the stateful core network is continuously updated when subscribers or the topology changes. This requires significant signaling effort, in particular, when failures hit the network.

The IETF propoesed Bit Index Explicit Replication (BIER) [WRD+17] as an efficient and scalable transport mechanism for multicast traffic. In Merling et al. [MMWE18] we give an overview and tutorial on BIER. At this point we briefly summarize the content.

BIER is a domain-based, point-to-multipoint forwarding mechanism for IPMC traffic. In contrast to traditional IPMC, it keeps the core network stateless, i.e., its core routers do not maintain information that depends on the subscribers. Figure 2.9 shows the layered architecture of BIER. When an IPMC packet enters the BIER domain at a

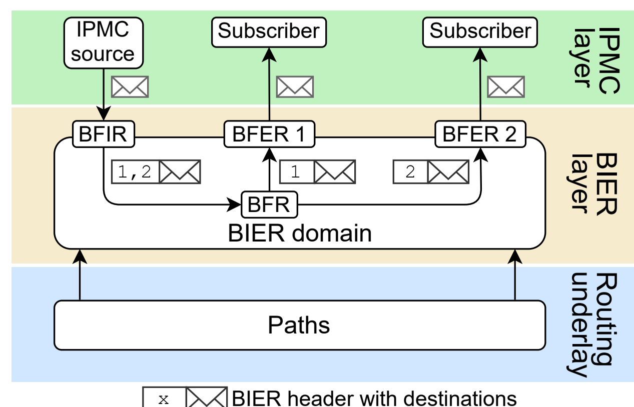


Figure 2.9: Layered BIER architecture (figure from Merling et al. [MLM21a]).

bit forwarding ingress router (BFIR), it is equipped with a BIER header. The BIER header contains all destinations of the packet. That is, it contains a bitstring where each destination in the BIER domain, i.e., bit forwarding egress router (BFER), is assigned a position. If a specific BFER should receive a packet copy, its corresponding bit is activated. In the BIER domain, bit forwarding routers (BFRs) replicate and forward BIER packets towards all destinations of the BIER packet, i.e., BFERs, indicated in its BIER header. They do so along paths from the unicast routing, i.e, the Interior Gateway Protocol (IGP) which we call routing underlay in the following. Before forwarding a packet to a particular neighboring BFR, the forwarding BFR clears the bits of BFERs that are reached via other neighboring BFRs in the bitstring in the BIER header. This

prevents duplicates at the receivers. Finally, BFERs remove the BIER header from the packet.

2.2.2 BIER Fast Reroute (BIER-FRR)

In Section 2.1 we described how to make unicast data plane traffic in SDN resilient. Failures affect multicast traffic, too. However, as multicast traffic is distributed along tree-like structures, a failure may disconnect entire subtrees, rendering many destinations unreachable.

In Merling et al. [MLM20a] we developed and compared two approaches to protect BIER traffic with fast reroute mechanism, i.e, BIER fast reroute (BIER-FRR). At this point we only briefly describe both approaches and refer to the original paper for more details [MLM20a].

The first BIER-FRR approach is based on LFAs. It reroutes BIER packets to alternative NHs around the failure so that BIER traffic can be successfully delivered. Thereby, the rerouting BFR adapts the bistrings in the BIER headers to avoid duplicates at receivers. The second approach leverages unicast protection tunnels to all next-next-hop BFRs. To that end, the rerouting BFR encapsulates copies of the BIER packet with unicast headers. The destinations of those temporary unicast packets are the next-next-hop BFRs downstream of the failure. Unicast routing usually has strong FRR capabilities or fast recomputation of forwarding rules so that packets are still delivered at the BFRs. At the BFRs the unicast headers are removed and the original BIER packets are forwarded.

2.2.3 BIER Scalability

BIER stores the destinations of a packet in the BIER header. Thus, the header size increases in larger networks. This rises questions with regard to the efficiency and scalability of BIER since large packet headers are inefficient. In Merling et al. [MSM22] we perform comprehensive evaluations to investigate the scalability of BIER. At this point we summarize the paper and present only the most important results. That is, we briefly explain how BIER is deployed in large networks and the resulting research objective. Finally, we present results of the study with regards to overhead of BIER, its traffic saving potentials and selection of reasonable BIER header size.

2.2.3.1 Subdomains

When BIER is deployed in a large network, the BIER domain is divided into multiple subdomains to avoid large BIER headers. That is, the BFERs are divided into sets with their own BIER headers. When an BFIR receives an IPMC packet, it sends a separate BIER packet into the BIER domain for each subdomain that contains at least one destination of the IPMC packet. This reduces the BIER header size to the maximum subdomain size but forwards multiple BIER packets into the BIER domain for each IPMC packet.

2.2.3.2 Research Objectives

In Merling et al. [MSM22] we evaluated the increase of traffic load in the network when BIER with subdomains is used. That is, when a BFIR sends the BIER packet copies to the subdomains, sometimes they are forwarded over the same link which is contrary to the idea of multicast and adds additional load on the links. We investigate the overhead of BIER with subdomains in comparison to traditional IPMC and how much traffic is saved by BIER in comparison to unicast. Furthermore, we evaluate the effect of the bitstring size in the BIER header since the BIER RFC [WRD$^+$17] suggests a range for that size but makes no comments on the effects and efficiency.

2.2.3.3 Clustering Algorithms

The assignment of BFERs to subdomains heavily influences the overall traffic load within the BIER domain. Therefore, we briefly discuss the aspect of clustering a BIER topology into subdomains. Please refer to Merling et al. [MSM22] for details.

The BIER RFC [WRD$^+$17] does not specify any clustering algorithm. In Merling et al. [MSM22] we present an integer linear program (ILP) that computes an optimal assignment so that subdomains are of equal size[14], and that the overall traffic load is minimized. However, the ILP is solvable in reasonable time only in small topologies, i.e., in moderate-size and large topologies no solution was found even after several hours.

Therefore, we described an heuristic to approximate the solution of the ILP. The basic idea is to initially chose n random subdomain centers in the topology and then assign

[14]This prevents subdomains with extensive size, and therefore, large BIER headers.

BFERs in a round robin manner to the nearest subdomain center whose subdomain is not full, yet. Afterwards, BFERs are swapped between subdomains to improve the assignment.

We compared the heuristic to ILP solutions in small networks and showed that the heuristically calculated results differ less then 1% from the optimum[15]. Therefore, we present evaluations based on the heuristic clustering in the following. Furthermore, we designed optimal cluster algorithms for selected topologies, i.e., full-mesh, ring, binary-tree, and line, which we use for those topologies.

2.2.3.4 Overall Traffic in Networks of Different Sizes

In this section we evaluate the overall traffic amount in different topologies and network sizes. First, we explain the methodology. Then, we present results for BIER compared to traditional IPMC, and BIER compared to unicast. Finally, we discuss the results.

2.2.3.4.1 Methodology We evaluate selected network types, i.e., full-mesh, ring, binary-tree, line, and randomly generated mesh-d[16] $d \in \{2, 4, 6, 8\}$ networks. We consider networks of sizes $k \in \{256, 512, 1024, 2048, 4096, 8192\}$. Every node is a BFIR, BFR, and BFER. We cluster the topologies into $n \in \{1, 2, 4, 8, 16, 32\}$ subdomains of sizes 256, i.e., the default BIER bitstring length, and send an IPMC packet from any node in the network to all other nodes.

We evaluate the overall traffic load which is the number of hops that are required to distribute all packet. We separately report results of overall traffic load of BIER relative to both traditional IPMC and unicast. This allows to assess the overhead of BIER in comparison to traditional IPMC, and the traffic saving potentials of BIER compared to unicast.

We repeat all evaluations 10 times and report average results. We computed 95% confidence intervals but omit them in the figures because they were very small.

2.2.3.4.2 BIER Overhead Figure 2.10 presents the overall traffic load of BIER relative to IPMC. In general, we see that BIER increases the traffic volume in comparison to IPMC, i.e., the relative traffic load is always greater or equal to one. This is not

[15] See Section V.C. of Merling et al. [MSM22] for details of the evaluation.
[16] In a mesh-d network the average node degree is d.

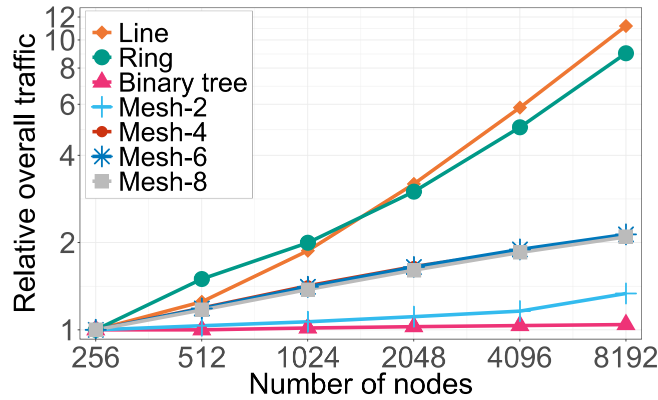


Figure 2.10: BIER has a higher overall traffic load than IPMC(figure from Merling et al. [MSM22]).

surprising because for every IPMC packet BIER sends an individual BIER packet to each subdomain. In binary-trees, and mesh-2 topologies the traffic load increase is always below 40% although it is often significantly smaller. For mesh-4, mesh-6, mesh-8 the traffic volume rises linearly from no increase in topologies of size 256 to up to a doubling of the traffic amount in very large topologies with 8192 nodes. In very large line and ring topologies the traffic volume increases by far the most to up to 12 times the amount compared to IPMC.

2.2.3.4.3 BIER Traffic Saving Potentials Figure 2.11 shows the results for BIER in comparison to unicast. In general, BIER decreases the traffic volume in comparison

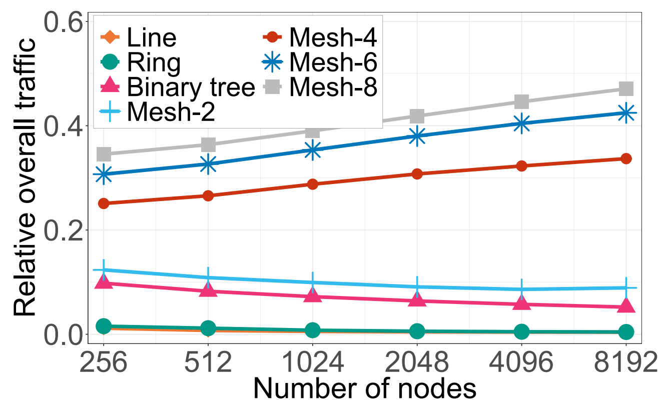


Figure 2.11: BIER has a significantly lower traffic load then unicast (figure from Merling et al. [MSM22]).

to unicast significantly. The effect ranges from almost 99% reduction in line and ring topologies to around 50% decrease in very large mesh-8 topologies. The figure shows the interesting effect that the relative overall traffic of BIER compared to unicast increases with growing network size. We explain this effect by the increasing number of required subdomains in larger networks. Therefore, more packets are sent for each IPMC packet, which increases the overall traffic.

2.2.3.4.4 Discussion The results show that in terms of traffic volume BIER is a compromise between IPMC which sends only one packet per involved link and unicast which sends one individual packet per destination. BIER causes more traffic than IPMC, which is not surprising. However, the increase is reasonable, especially because IPMC comes with all the disadvantages, described in Section 1.3, which lead to the development of BIER in the first place. Still, BIER significantly reduces the traffic amount in comparison to unicast in all topologies and network sizes.

2.2.3.5 BIER Header Size

Now, we evaluate the effect of the BIER header size. That is, large bitstrings in the BIER header potentially reduce the number of required subdomains. However, a large header also increase the traffic amount in the network that is not payload. Therefore, we measure the absolute overall traffic that is sent through a BIER domain in a network with 8192 nodes with different BIER header sizes. We assume the payload to be 500 bytes, i.e., the average packet size in the internet [L^+12]. We omit mesh-6 and mesh-8 topologies because results were very similar to mesh-2 and mesh-4. Figure 2.12 shows the results. We see different effects for line and ring topologies, and all other topologies. In line and ring topologies the absolute overall traffic reduces with increasing header size up around 2048 bits from which it stays almost the same for larger headers. That is because at this point the benefit of fewer subdomains is smaller than the larger header overhead. For all other topologies the traffic volume slowly grows with increasing header size. Therefore, we argue that for such topologies the optimal bitstring size is between 256 and 1024 bits.

In general, we argue that the optimal bitstring size depends on the topology. In particular, the average shortest path lengths, highly influences the efficiency. Topologies with long average shortest path length, i.e., line and ring topologies, benefit from larger

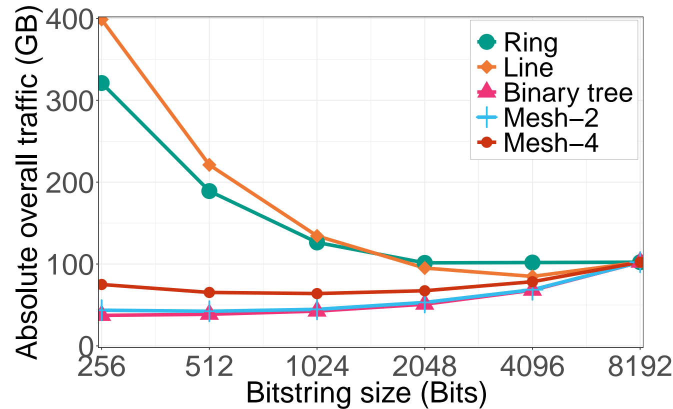


Figure 2.12: Effect ofBIERheader size on the overall traffic volume (figure from Merling et al. [MSM22]).

headers because it is cheaper to forward one large packet along a long path then to forward multiple small packets. In contrast, topologies with shorter average shortest paths benefit from smaller header sizes.

2.2.4 BIER Implementation in P4

In this section we present the most important results from Merling et al. [MLM20b] and Merling et al. [MLM21a]. We implemented BIER and BIER-FRR for the BMv2 [MLM20b] as a proof of concept, and for the P4-programmable high-performance switching ASIC Tofino [MLM21a]. In this summary, we focus on the Tofino implementation because it provides more relevant insights and interesting results for researches and developers. We discuss the implementation concepts, present hardware-based evaluation results, and introduce means to increase the efficiency of the implementations.

2.2.4.1 Implementation Concept

Figure 2.13 shows the concept of the BIER implementation in P4. The Tofino has two processing pipelines, i.e., the ingress and the egress. BIER packets are first processed in the ingress where a next-hop (NH) of the BIER packet is determined. Afterwards the packet is cloned so that two packets enter the egress pipeline. The original BIER packet is sent towards the earlier determined NH. The packet copy is recirculated. That is, the packet is again processed by the ingress as if it has been received regularly.

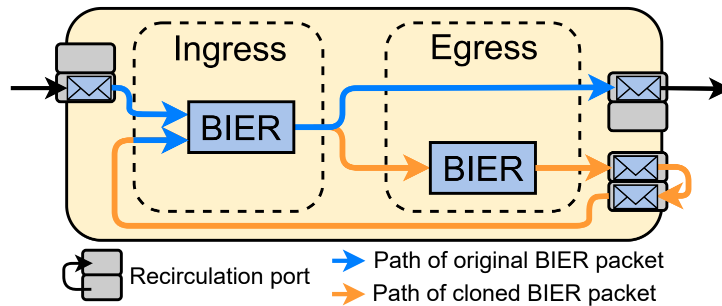


Figure 2.13: BIER packet flow in the P4 processing pipeline (figure from Merling et al. [MLM21a]).

This procedure is repeated until the BIER packet has been forwarded to all its NHs. Therefore, a BIER packet with n NHs is recirculated $n - 1$ times.

The downside of this concept is that packets are cloned and processed again, which requires additional processing capacity[17]. For example when packets arrive with 100 Gb/s at a switch and are recirculated once, 200 Gb/s processing capacity is required. Then, packets may be dropped because usually the recirculation capacity of the Tofino is limited 100 Gb/s. Therefore, we introduce so-called recirculation ports. Such a port is a physical port of the switch that is configured to provide 100 Gb/s additional processing capacity for recirculation traffic. However, such ports cannot be used for regular forwarding.

To facilitate readability, we refer the build-in capacity of 100 Gb/s for recirculation traffic of the Tofino also as a recirculation port although it does not occupy a physical port. It should be used before any physical ports are configured as recirculation ports.

2.2.4.2 Evaluations

In this section we evaluate the throughput of the BIER P4 implementation. Furthermore, we predict throughput results for realistic traffic mixes.

2.2.4.2.1 Throughput Measurements At this point we omit details of the hardware setup and refer to Section VI.2) of Merling et al. [MLM21a]. We send BIER packet with 100 Gb/s with multiple NHs to the Tofino. Thereby, we configured the Tofino with different numbers of recirculation ports, i.e., varying recirculation capacity. We report

[17]Currently, P4 does not support more efficient dynamic packet cloning mechanisms.

the end-to-end throughput at the first receiver[18]. Figure 2.14 shows the results. We see

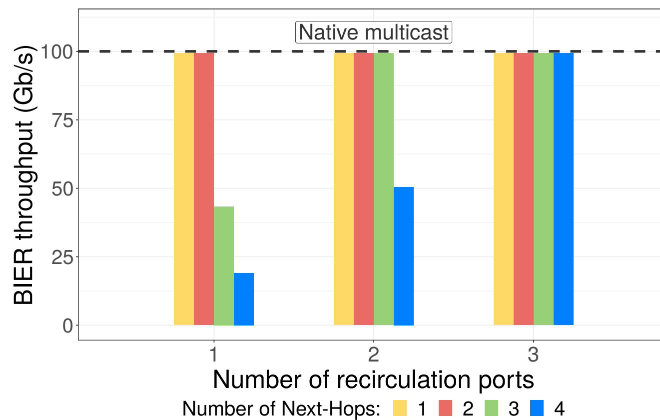


Figure 2.14: The throughput depends on the available recirculation capacity and number of NHs (figure from Merling et al. [MLM21a]).

that end-to-end throughput is 100 Gb/s with only one recirculation port when the BIER packet has up to two NHs. That is because the packets are recirculated at most once. When an additional NH is added the throughput drops to around 45 Gb/s and even further when a fourth NH is added. That is because one recirculation port provides insufficient capacity for recirculated traffic, and therefore, packets are dropped. When more recirculation ports are added, more recirculation capacity is available so that the throughput is not negatively affected even when the number of NHs increases.

2.2.4.2.2 Throughput Predictions for Realistic Traffic Mixes We performed the hardware-based evaluations with 100 Gb/s traffic rate to show the capabilities of the BIER implementation. However, normally multicast makes up only a small fraction of the total traffic compared to unicast. Therefore, we vary the fraction $a \in \{1, 2.5, 5, 10\}\%$ of multicast traffic of the the total traffic of 100 Gb/s per link. In addition, we vary the number of NHs[19]. Then, we perform simulations to calculate the number of required recirculation ports to prevent packet loss. Figure 2.15 shows the results. In networks with 2.5% multicast traffic or less, 2 recirculation ports provide enough capacity for BIER packets with up to 16 NHs. For larger traffic fractions the number is still rather low if reasonable numbers of NHs are considered. In general, we conclude that in realistic traffic mixes few recirculation ports suffice to prevent packet loss.

[18]With unlimited recirculation capacity this should always be 100 Gb/s.
[19]This influences the number of recirculations per packet.

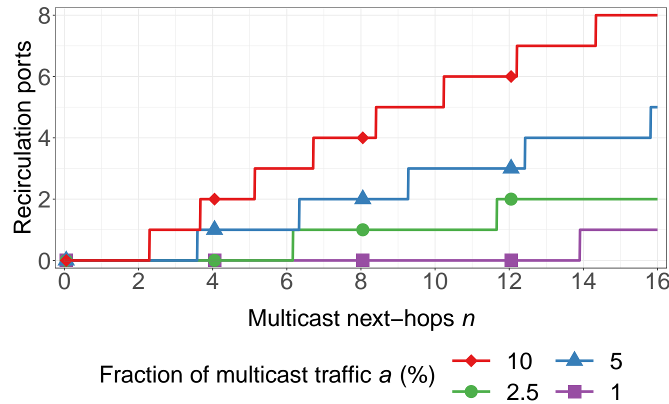


Figure 2.15: Number of required recirculation ports to prevent packet loss (figure from Merling et al. [MLM21a]).

2.2.4.3 Efficient BIER Implementation

In Lindner et al. [LMM22] we suggest ways to make BIER forwarding in a switch more efficient. As I am only a contributing author to that work I only briefly summarize the paper in the following.

We propose the following forwarding mechanism to improve BIER forwarding. The switch observes to which egress ports it sends BIER packets. Over time, it identifies ports that frequently occur together in a packet, i.e., egress port sets. Then, the controller installs static multicast groups on the switch for the observed egress port sets[20]. Such static multicast groups forward BIER packets over several egress ports simultaneously without packet recirculation. Thus, this approach sends a BIER packet to multiple NHs per pipeline iteration instead of only one NH. This significantly reduces the number of recirculations per packet, and therefore, the required recirculation capacity. The switch continuously observes the egress port sets and adapts the static multicast groups over time if necessary.

2.2.5 Discussion and Outlook

The presented P4 BIER prototype enables BIER forwarding in P4. Furthermore, the evaluations show that BIER is a scalable transport mechanism for IPMC that does not

[20]Installing static multicast groups for all possible egress port sets would process any BIER packet without packet recirculation. However, this is unfeasible because it requires 2^{32} static multicast groups on a 32 port switch.

suffer the downsides of traditional multicast approaches. However, the implemented BIER packet processing approach requires packet recirculation, and therefore, additional recirculation capacity, to prevent packet loss under heavy load. Future research could focus on making BIER forwarding in P4 more efficient. In particular, BIER forwarding that processes all NHs of a BIER packet in one pipeline iteration would be desirable.

Currently, the IETF is developing an alternative encoding for multicast trees in the packet header [Tro22] which is more efficient in large networks with low number of multicast subscribers. Future research could investigate this mechanism, and assess its feasibility in P4, and evaluate its efficiency.

Visibility of the Results

Merling et al. [MLM20b] (see Appendix 1.5) contains a detailed implementation of BIER and BIER-FRR for a P4 software switch to show how complex forwarding behavior is implemented in P4. It was published in the Journal of Network and Computer Applications in 2020, the source code is publicly available[21], and the results were extensively discussed in multiple IETF BIER working group meetings. Furthermore, the work supported the development of a BIER-FRR draft [MM19] in the BIER working group and became an active working group document [CML^{+}22].

Merling et al. [MLM21a] (see Appendix 1.6) contains interesting evaluations of the performance of BIER and BIER-FRR in a high-performance hardware testbed. The paper was published as open access in the IEEE Access journal and the source code is publicly available[22]. The implementation and the results were heavily discussed in several IETF BIER working group meetings and several other smaller workshops.

Merling et al. [MSM22] (see Appendix 1.7) is a comprehensive evaluation of the scalability and efficiency of BIER. It has been accepted for publication in the IEEE Transactions on Network and Service Management journal. The results will be presented and discussed in the IETF BIER working group.

Lindner et al. [LMM22] (see Appendix 1.8) describes efforts to make BIER forwarding more efficient in P4. It has been accepted for publication in the „Machine Learning and

[21] https://github.com/uni-tue-kn/p4-bier
[22] https://github.com/uni-tue-kn/p4-bier-tofino

Artificial Intelligence for Managing Networks, Systems, and Services" special issue of the IEEE Transactions on Network and Service Management journal.

Finally, Merling et al. [MMWE18] (see Appendix 1.3) was published at the IETF Journal in 2018 and has been used as an overview and introduction to BIER.

2.3 Additional Content

Research in this section is only additional content of this thesis and I have been only contributing author. Therefore I present the work only briefly.

2.3.1 P4 ABC

P4 ABC [MMMH19][23] is the P4 implementation of activity-based congestion management (ABC) [MZ16]. ABC is a mechanism to ensure fairness between multiple senders in a domain which does not require per-user or per-flow state in its core. To that end, ingress nodes measure the activity of users and flows and annotate packets with activity information. Nodes in the core drop packets when congestion is imminent based on the activity value in the packet header where packets with higher activity experience a drop more likely.

We implemented P4 ABC for the P4 software switch BMv2 and performed evaluations. The results show that without ABC an user with high activity, i.e., high constant bitrate (CBR) traffic, is able to push away a user that sends TCP traffic which decreases its sending rate to prevent congestion. When ABC is activated, the TCP user gets a significantly larger share of bandwith because ABC drops packets of the CBR user more likely due to the higher activity.

2.3.2 Load Profile Negotiation

In Heimgärtner et al. [HHS+19][24] we presented a scheduling algorithm to find cost-optimized load schedules for energy consumers based on day-ahead forecasts. To that end, users register their schedules at an aggregator which trades energy at the spot market on behalf of its users. The aggregator checks whether the provided schedules

[23] published on the Future Internet Journal by MDPI

[24] International ETG-Congress 2019, ETG Symposium

conflict each other, i.e., whether the accumulated power demands in any timeslot can be met. If so, it renegotiates schedules with the users in a way that those constraints are met and energy cost is reduced by shifting flexibilities to cheaper time slots in the day-ahead price forecast. The results show the feasabiliy of this approach. However, due to the simple model, results cannot be generalized and further research is needed.

Bibliography

[AZ08] A. Atlas and A. Zinin. RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates. Internet-draft, Internet Engineering Task Force, September 2008. http://www.rfc-editor.org/rfc/rfc5286.txt.

[Bas18] Antonin Bas. BMv2 Throughput. https://github.com/p4lang/behavioral-model/issues/537#issuecomment-360537441, January 2018. Accessed: 22 April 2023.

[BDG+14] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communications Review (CCR)*, 44, 2014.

[BFP+15] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So. RFC7490: Remote Loop-Free Alternate (LFA) Fast Reroute (FRR). Internet-draft, Internet Engineering Task Force, April 2015. https://tools.ietf.org/html/rfc7490.

[BM16] Wolfgang Braun and Michael Menth. Loop-Free Alternates with Loop Detection for Fast Reroute in Software-Defined Carrier and Data Center Networks. *Journal of Network and Systems Management*, 24:470–490, 2016.

[CKR+21] Marco Chiesa, Andrzej Kamisiński, Jacek Rak, Gábor Rétvári, and Stefan Schmid. A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks. *IEEE Communications Surveys & Tutorials*, 23:1253–1301, 2021.

[CML+22] Huaimo Chen, Mike McBride, Steffen Lindner, Michael Menth, Aijun Wang, Gyan Mishra, Yisong Liu, Yanhe Fan, Lei Liu, and Xufeng Liu. BIER Fast ReRoute. Internet-draft, Internet Engineering Task

Force, April 2022. https://datatracker.ietf.org/doc/draft-chen-bier-frr/05/.

[CR15] L. Csikor and G. Retvari. On Providing Fast Protection with Remote Loop-Free Alternates: Analyzing and Optimizing Unit Cost Networks. *Telecommunication Systems*, 60:485–502, 2015.

[dSMSF15] Anderson Santos da Silva, Paul Smith, Andreas Mauthe, and Alberto Schaeffer-Filho. Resilience support in software-defined networking: A survey. *Computer Networks*, 92:189–207, 2015.

[Edg17] Edge-Core Networks. Wedge100BF-32X/65X Switch. https://www.edge-core.com/_upload/images/Wedge_100-32X_DS_R04_20170615.pdf, 2017. Accessed: 22 April 2023.

[EXF19] EXFO. FTB-1v2/FTB-1 Pro Platform. https://www.exfo.com/umbraco/surface/file/download/?ni=10900&cn=en-US&pi=5404, 2019. Accessed: 22 April 2023.

[FB17] Adrian Farrel and Ron Bonica. Segment Routing: Cutting Through the Hype and Finding the IETF's Innovative Nugget of Gold. *IETF Journal*, 13, 2017.

[HHM21] Frederik Hauser, Marco Häberle, and Michael Menth. P4sec: Automated Deployment of 802.1X, IPsec, and MACsec Network Protection in P4-Based SDN. Technical report, University of Tuebingen, 2021.

[HHM+23] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research. *Journal of Network and Computer Applications*, 212, 2023.

[HHS+19] Florian Heimgaertner, Sascha Heider, Thomas Stueber, Daniel Merling, and Michael Menth. Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning. In *International ETG-Congress 2019; ETG Symposium*, pages 1–6, 2019. ©2019 IEEE. Reprinted with permission.

[HHSM20] Frederik Hauser, Marco Häberle, Marc Schmidt, and Michael Menth. P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN. *IEEE Access*, 8:139567–139586, 2020.

[Int22] Intel®. Intel® Tofino™. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, 2022. Accessed: 22 April 2023.

[KNF+11] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29:1765–1775, 2011.

[L+12] F. Liu et al. The packet size distribution patterns of the typical internet applications. In *IEEE International Conference on Network Infrastructure and Digital Content*, pages 325–332, 2012.

[LBF+22] Stephane Litkowski, Ahmed Bashandy, Clarence Filsfils, Pierre Francois, Bruno Decraene, and Daniel Voyer. Topology Independent Fast Reroute using Segment Routing. Internet-draft, Internet Engineering Task Force, January 2022. https://datatracker.ietf.org/doc/draft-ietf-rtgwg-segment-routing-ti-lfa/08/.

[LMHM20] Steffen Lindner, Daniel Merling, Marco Häberle, and Michael Menth. P4-Protect: 1+1 Path Protection for P4. In *Proceedings of the 3rd P4 Workshop in Europe*, page 21–27, 2020. https://doi.org/10.1145/3426744.3431327.

[LMM22] Steffen Lindner, Daniel Merling, and Menth Michael. Learning Multicast Patterns for Efficient BIER Forwarding with P4, 2022. Accepted for publication in IEEE Transactions on Network and Service Management (TNSM) journal. The most recent version of this publication can be found in the Appendix 2.2.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communications Review (CCR)*, 38:69–74, March 2008.

[MBM18] Daniel Merling, Wolfgang Braun, and Michael Menth. Efficient Data Plane Protection for SDN. In *IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 10–18, 2018.

[Mer17] Daniel Merling. Scalable Resilience for Software-Defined Networking using Remote Loop-Free Alternates with Loop Detection. Master's thesis, University of Tuebingen, Sand 13, 72076 Tübingen, September 2017.

[MLM20a] Daniel Merling, Steffen Lindner, and Michael Menth. Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast. In *Proceedings of Seventh International Conference on Software Defined Systems (SDS)*, pages 51–58, 2020. ©2020 IEEE. Reprinted with permission.

[MLM20b] Daniel Merling, Steffen Lindner, and Michael Menth. P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast. *Journal of Network and Computer Applications*, 169:102764, 2020.

[MLM21a] Daniel Merling, Steffen Lindner, and Michael Menth. Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4. *IEEE Access*, 9:34500–34514, 2021. ©2021 IEEE. Reprinted with permission.

[MLM21b] Daniel Merling, Steffen Lindner, and Michael Menth. Robust LFA Protection for Software-Defined Networks (RoLPS). *IEEE Transactions on Network and Service Management (TNSM)*, 18:2570–2586, 2021. ©2021 IEEE. Reprinted with permission.

[MM19] Daniel Merling and Michael Menth. BIER Fast Reroute. Internet-draft, Internet Engineering Task Force, March 2019. `https://datatracker.ietf.org/doc/draft-merling-bier-frr/00/`.

[MMMH19] Michael Menth, Habib Mostafaei, Daniel Merling, and Marco Häberle. Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC). *Future Internet*, 11, 2019. ©2019 IEEE. Reprinted with permission.

[MMWE18] Daniel Merling, Michael Menth, Nils Warnke, and Toerless. Eckert. An Overview of Bit Index Explicit Replication (BIER). In *IETF Journal*, 2018.

[MSM22] Daniel Merling, Thomas Stüber, and Michael Menth. Efficiency of BIER Multicast in Large Networks, 2022. Accepted for publication in IEEE Transactions on Network and Service Management (TNSM) journal. The most recent version of this publication can be found in the Appendix 2.1.

[MZ16] Michael Menth and Nikolas Zeitler. Activity-Based Congestion Management for Fair Bandwidth Sharing in Trusted Packet Networks. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 231–239, 2016.

[p4l19] p4lang. behavioral-model. `https://github.com/p4lang/behavioral-model`, 2019. Accessed: 22 April 2023.

[PSPM17] Jozef Papan, Pavel Segeč, Peter Palúch, and Ludovit Mikus. The Survey of Current IPFRR Mechanisms. In *Federated Conference on Software Development and Object Technologies*, pages 229–240, December 2017.

[RI07] A. Raj and O.C. Ibe. A Survey of IP and Multiprotocol Label Switching Fast Reroute Schemes. *Computer Networks*, 51:1882–1907, 2007.

[RMD05] Smita Rai, Biswanath Mukherjee, and Omkar Deshpande. IP Resilience within an Autonomous System: Current Approaches, Challenges, and Future Directions. *IEEE Communications Magazine*, 43:142–149, 2005.

[SHB+17] P. Sarkar, S. Hegde, C. Bowers, H. Gredler, and S. Litkowski. RFC8102: Remote-LFA Node Protection and Manageability. Internet-draft, Internet Engineering Task Force, March 2017. `https://tools.ietf.org/html/rfc8102`.

[Tro22] Dirk Trossen. Forward Requests Return Multicast (FRRM) Communication Semantic. Internet-draft, Internet Engineering Task Force, July 2022. `https://datatracker.ietf.org/doc/draft-trossen-rtgwg-frrm/00/`.

[WRD+17] IJsbrand Wijnands, Eric C. Rosen, Andrew Dolganow, Tony Przygienda, and Sam Aldrin. RFC8279: Multicast Using Bit Index Explicit Replication (BIER). Internet-draft, Internet Engineering Task Force, November 2017. `https://datatracker.ietf.org/doc/rfc8279/`.

Personal Contribution

Accepted Manuscripts (Core Content)

1. **Robust LFA Protection for Software-Defined Networks (RoLPS)** [MLM21b]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)" funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was the development and implementation of a scalable and efficient 1:1 resilience mechanism for SDNs.
Names of collaborators and their shares	Irene Müller-Benz: Implementation of an early prototype in her master thesis. Steffen Lindner: Co-supervision of the master thesis of Irene Müller-Benz (focus on technological supervision). Implementation of the developed concepts. Editorial assistance on the publication. Michael Menth: Scientific supervision of the master thesis and research work. Editorial assistance on the publication.
Importance of own contributions to the joint work	Conceptual development of the protection mechanism. Supervision of the master thesis of Irene Müller-Benz (focus on concept and architecture). Main author of the publication taking on most of the write-up and rewriting during revision phase.

2. **P4-Protect: 1+1 Path Protection for P4** [LMHM20]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was the development and implementation of a scalable and efficient 1+1 resilience mechanism for SDNs.
Names of collaborators and their shares	Steffen Lindner: Main developer for design of resilience mechanism. Responsible for the Implementation. Main author of the publication. Marco Häberle: Editorial assistance on publication and implementation support. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Editorial assistance for writing of the publication and assistance during development and implementation phase.

3. **An Overview of Bit Index Explicit Replication (BIER)** [MMWE18]

Scope of the joint work	This work was done for the Internet Engineering Task Force (IETF). The goal was to write up a refined overview and tutorial for BIER for the IETF Journal.
Names of collaborators and their shares	Nils Warnke, Toerless Eckert: Technical input and co-author. Michael Menth: Scientific supervision, co-author and editorial assistance.
Importance of own contributions to the joint work	Main author of the paper, taking up most of the write-up.

4. **Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast** [MLM20a]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was the development and comparison of fast resilience mechanisms for BIER, i.e., BIER-FRR.
Names of collaborators and their shares	Steffen Lindner: Editorial assistance on the publication and assistance during development phase. Michael Menth: Scientific supervision. Editorial assistance on the publication.
Importance of own contributions to the joint work	Conceptual development of protection mechanism. Main author of the publication taking on most of the write-up and rewriting during revision phase.

5. **P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast** [MLM20b]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was the development of a fast protection mechanism for BIER, i.e., BIER-FRR, and implementing a BIER and BIER-FRR prototype for the P4 software switch BMv2.
Names of collaborators and their shares	Steffen Lindner: Implementation of the developed concepts. Editorial assistance and co-author of the publication. Michael Menth: Scientific supervision. Editorial assistance on the publication.
Importance of own contributions to the joint work	Mainly responsible for the developed concepts. Main author of the publication taking on most of the write-up and rewriting during revision phase.

6. **Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4** [MLM21a]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was the implementation and evaluation of a BIER and BIER-FRR prototype for high-performance P4-programmable hardware.
Names of collaborators and their shares	Steffen Lindner: Responsible for the implementation. Editorial assistance and co-author of the publication. Michael Menth: Scientific supervision. Editorial assistance on the publication.
Importance of own contributions to the joint work	Assistance on implementation. Main author of the publication taking on most of the write-up and rewriting during revision phase.

Submitted Manuscripts (Core Content)

7. **Efficiency of BIER Multicast in Large Networks** [MSM22]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was to design clustering algorithms for BIER subdomains and to evaluate the efficiency and scalability of BIER and BIER-FRR.
Names of collaborators and their shares	Tobias Burghardt, Alexander Rzehak, Manuel Eppler: Bachelor thesis in the context of this research work. Early results and insights. Thomas Stüber: Design of suitable algorithms. Co-supervisor of bachelor theses. Editorial assistance and co-author of the publication Michael Menth: Scientific supervision of the bachelor theses and publication. Editorial assistance on the publication.

Importance of own contributions to the joint work	Supervisor of bachelor theses. Supervision of design and evaluation of developed mechanisms and assistance. Main author of the publication taking on most of the write-up and rewriting during revision phase.

8. **Learning Multicast Patterns for Efficient BIER Forwarding with P4** [LMM22]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG). The scope of this work was to make BIER forwarding in P4 more efficient.
Names of collaborators and their shares	Steffen Lindner: Mainly responsible for concept development. Implementation of the concepts. Main author of the publication. Michael Menth: Scientific supervision. Editorial assistance on the publication.
Importance of own contributions to the joint work	Assistance on concept development. Co-author of and editorial assistance on publication.

9. **A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Re- search** [HHM+23]

Scope of the joint work	Literature study on the state-of-the-art of data plane programming with P4 covering fundamentals, advances, and applied research.

Names of collaborators and their shares	Frederik Hauser: Coordination of all activities around this writing project. Main author of the publication. Steffen Lindner, Marco Häberle: Analysis, classification and summarization of applied research works. Writing input and feedback on the foundation chapters. Help in structure definition and review of the complete manuscript. Vladimir Gurevich: Writing input and feedback on the foundation chapters. Florian Zeiger, Reinhard Frank: Feedback on the structure and write-up of the manuscript. Michael Menth: Scientific supervision of the research project. Editorial assistance on the publication.
Importance of own contributions to the joint work	Analysis, classification and summarization of applied research works. Writing input and feedback on the foundation chapters. Help in structure definition and review of the complete manuscript.

Accepted Manuscripts (Additional Content)

10. **Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)** [MMMH19]

Scope of the joint work	The scope of this work was the implementation of activity-based congestion management (ABC) for the BMv2 P4 software switch and its evaluation.

Names of collaborators and their shares	Michael Menth: Scientific supervision and main author of the publication. Habib Mostafaei: Responsible for the implementation. Editorial assistance and co-author of the publication. Marco Häberle: Implementation support and editorial assistance.
Importance of own contributions to the joint work	Editorial assistance and implementation support.

11. **Load Profile Negotiation in Day-Ahead Planning for Compliance with Power Limits** [HHS+19]

Scope of the joint work	The scope of this work was the development of a mechanism to find cost-optimized load schedules for energy consumers based on day-ahead forecast.
Names of collaborators and their shares	Florian Heimgärtner: Development of concepts. Scientific supervision of the bachelor thesis and main author of the publication. Sascha Heider: Bachelor thesis on the design of a suitable mechanism. Thomas Stüber: Scientific supervision of the bachelor thesis and editorial support. Michael Menth: Supervision of the bachelor thesis and editorial assistance.
Importance of own contributions to the joint work	Supervision of the bachelor thesis and editorial assistance during write up.

Publications

1 Accepted Manuscripts (Core Content)

1.1 Robust LFA Protection for Software-Defined Networks (RoLPS)

©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes,creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Robust LFA Protection for Software-Defined Networks (RoLPS)

Daniel Merling, Steffen Lindner, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Germany
{daniel.merling, steffen.lindner, menth}@uni-tuebingen.de

***Abstract*—In software-defined networks, forwarding entries on switches are configured by a controller. In case of an unreachable next-hop, traffic is dropped until forwarding entries are updated, which takes significant time. Therefore, fast reroute (FRR) mechanisms are needed to forward affected traffic over alternate paths in the meantime. Loop-free alternates (LFAs) and remote LFAs (rLFAs) have been proposed for FRR in IP networks. However, they cannot protect traffic for all destinations and some LFAs may create loops under challenging conditions.**

This paper proposes robust LFA protection for software-defined networks (RoLPS). RoLPS augments the coverage of (r)LFAs with novel explicit LFAs (eLFAs). RoLPS ranks available LFAs according to protection quality and complexity for selection of the best available LFA. Furthermore, we introduce advanced loop detection (ALD) so that RoLPS stops loops caused by LFAs. We evaluate RoLPS-based protection variants on a large set of representative networks with unit and non-unit link costs. We study their protection coverage, additional forwarding entries, and path extensions for rerouted traffic, and compare them with MPLS facility backup. Results show that RoLPS can protect traffic against all single link or node failures, and against most double failures while inducing only little overhead. We implement FRR on the P4-programmable switch ASIC Tofino and provide a control plane logic based on RoLPS. Measurement results show that the prototype achieves a throughput of 100 Gb/s, reroutes traffic within less than a millisecond, and reliably detects and drops looping traffic.

***Index Terms*—Software-Defined Networking, P4, Loop-Free Alternates, Resilience, Link Protection, Node Protection, Scalability,**

I. Introduction

Software-defined networking (SDN) separates data plane and control plane of forwarding nodes. A controller computes and installs forwarding rules on data plane devices to instruct them how to process data packets. Packet forwarding is impaired when a next-hop becomes unreachable due to a failure, i.e., a failed link or a failed node. Without controller interaction, switches drop affected packets. However, notification of the controller, recomputation of forwarding rules, and their installation on data plane devices takes a considerable amount of time. This outage time is too long, in particular for the transport of realtime traffic.

In IP networks fast reroute (FRR) mechanisms are used to quickly reroute packets via pre-computed backup paths while forwarding entries are recomputed. FRR would also be helpful in SDN to forward traffic with unreachable next-hops without controller interaction via alternate paths. However, SDN forwarding devices often have limited forwarding tables so that adding many forwarding entries for FRR purposes may be problematic. Loop-free alternates (LFAs) are a well-known FRR method for IP networks that requires no additional forwarding entries so that we consider them in this work. LFAs constitute alternative next-hops that successfully forward traffic towards the destination when the default next-hop is unreachable. The authors of [1] proposed to use LFAs to protect traffic without controller interaction in SDN-based networks. However, LFAs suffer from two major shortcomings. First, they cannot protect traffic for all destinations against single link failures (SLF) and single node failures (SNF). Second, some LFAs may cause rerouting loops in case of node failures or multiple failures.

In previous work [2] we improved the usage of LFAs in software-defined networks. We introduced explicit LFAs (eLFAs) based on explicit tunnels to protect destinations that cannot be protected by other LFAs. We proposed advanced loop detection (ALD) to detect and stop loops, which prevents severe overload that may happen with LFAs in failure cases. We described loop avoidance (LA), which leverages ALD, ranks available LFAs according to their protection quality and overhead, and chooses the best one. Furthermore, we showed how LA can be implemented in OpenFlow. Finally, a simulation-based evaluation showed that LA can protect all traffic in SDN networks against SLF and SNF and with less overhead compared to other FRR methods.

his paper is an extension of [2] with the following advances. (1) We augment eLFAs with explicit multipoint-to-point rerouting tunnels. This significantly decreases the required number of additional forwarding entries for explicit tunnels. (2) We modify ALD so that it can detect and stop loops faster while being implementable on P4 devices. (3) We update the simulative evaluations according to the new mechanisms. (4) We include topology-independent LFAs (TI-LFAs) [3] in the simulative evaluations because they are conceptually similar to eLFAs. (5) We improved the overall presentation, including a renaming of LA into RoLPS as the name LA did not capture the entire concept. (6) We implement a prototype of RoLPS on the P4-programmable switching ASIC Tofino featuring LFAs, rLFA, eLFA, and ALD, and a RoLPS-based SDN controller, and thereby, show its technical feasibility. (7) We demonstrate that the technical solution performs well by showing that the prototype operates at 100 Gb/s, reroutes traffic within less than

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

a millisecond, and reliably detects and drops looping traffic.

The paper is structured as follows. In Section II we discuss related work. Then, we review state of the art for LFAs in Section III. Section IV introduces eLFAs and ALD for improved protection of the SDN data plane, and a RoLPS-based control plane logic for that features and existing LFAs. Section V describes the simulative evaluation methodology and discusses performance results based on comprehensive study. We present the implementation of a P4-based hardware prototype in Section VI. We evaluate its performance in Section VII by measurements. Finally, we conclude the paper in Section VIII. A table of acronyms and a glossary are provided at the end of the paper to facilitate the reading.

II. Related Work

In this section we describe related work. First, we discuss legacy FRR mechanisms to position LFAs. Then, we review FRR for SDN.

A. FRR in Legacy Networks

Rai et al. [4], Raj et al. [5], and Papan et al. [6] present surveys that provide a wide overview of FRR in legacy networks. Hutchinson et al. [7] discuss the architecture and design of resilient network systems, i.e., specifying and realizing appropriate components. They review state-of-the-art contributions and identify future research issues.

1) MPLS Networks: For MPLS [8] two major FRR mechanisms have been proposed [9]. One-to-one backup reroutes packets on preconfigured paths that avoid the failure. Facility backup tunnels the packets locally around the failure to the next-hop for link protection, or to the next-next-hop for node protection. Only recently, the authors of [10] propose a loop detection mechanism for MPLS. It is based on special MPLS labels that are pushed on the MPLS header stack when a packet is rerouted. This allows nodes to detect whether a packet has already been rerouted.

2) IP Networks: Not-via addresses [11] protect both IP and MPLS networks. The routing table of a node contains one additional forwarding entry for every outgoing link. When the default next-hop is unreachable, those additional entries are used to deviate the packet from its shortest path through a tunnel around the failure. This causes a similar path layout as MPLS facility backup [12]. Failure insensitive routing (FIR) [13] leverages interface-specific routing tables to encode failure information. Depending on the ingress interface, packets are rerouted on precomputed backup paths around the failure. Multiple routing configurations (MRCs) [14] implement multiple disjoint routing topologies so that always at least one topology provides a working path towards the destination despite the failure. For each topology, an entire set of forwarding entries is required which at least doubles the amount of forwarding entries. Maximally redundant trees (MRTs) [15] leverage a similar approach. A red and a blue set of backup forwarding entries are computed so that at least one set delivers the packet in case of a failure. However, MRTs triple the number of forwarding entries in the network and may lead to extensive backup paths [16]. LFAs can be combined with MRTs to reduce backup path length and link load [17]. Independent directed acyclic graphs (IDAGs) [18] compute only two sets of maximally disjoint forwarding entries, i.e., doubling the amount of forwarding entries so that one is working in case of a failure. The authors of [19] encode failure information in the packet header. Nodes leverage this information to identify the failure and reroute packets on disjoint paths around it.

3) LFA-Based Protection: LFAs [20] with either link or node protection locally reroute packets around the failure on shortest paths. Therefore, they do not require additional forwarding entries but cannot protect all destinations. Csikor et al. [21], [22] increase the number of protected destinations by optimizing link costs. rLFAs [23]–[25] augment LFAs to increase the number of protected destinations by rerouting packets to remote nodes through shortest path tunnels. They do not need additional forwarding entries but still cannot protect all destinations. The performance of both LFAs and rLFAs can be enhanced by adding links to the network [26]. In [27], the authors present a self-configuring extension for LFAs based on probes. It installs alternative hops in other nodes to prevent rerouting loops. Topology-independent LFAs (TI-LFAs) [3] leverage segment routing (SR) [28] to protect against failures. SR is based on forwarding instructions in the packet header which may be stacked. TI-LFAs leverage SR to implement explicit tunnels to remote nodes. As eLFAs leverage explicit tunnels, too, they can be viewed as a very specific but rather untypical form of TI-LFAs.

B. FRR Protection in SDN

We discuss FRR in the context of SDN. We first address general FRR approaches for SDN and then we discuss related work for FRR in OpenFlow- and P4-based networks.

1) FRR in SDN: There have been many proposals to make the SDN control plane more resilient [29]. However, there are only very few efforts to protect traffic in the data plane. If the controller is notified about the failure, it may update its topology, and recompute and install updated forwarding entries. Sharma et al. [30] measure that recomputation takes about 80-100 ms. However, the authors clarify that this number highly depends on the number of affected flows, path lengths, and traffic bursts in the control network. In particular, it is likely that the time for rerouting is significantly higher in larger networks. Da Silva et al. [31] and Chiesa et al. [32] present surveys that give overviews of FRR in SDN with significantly faster protection than recomputation of forwarding entries.

2) OpenFlow-Based FRR: FRR capabilities have been introduced in OpenFlow with Version 1.1. The authors of [33] provide a BFD-based protection scheme for earlier OpenFlow versions than 1.1. It is based on a bidirectional forwarding detection (BFD) where nodes periodically exchange information about their reachability. Van Adrichem et al. [34] measure that failure detection takes about 3-30 ms on the software-based Open vSwitch depending on the configuration of the BFD. SlickFlow [35] encodes primary and backup paths in the packet header to reroute packets when an unavailable egress port is selected. SPIDER [36] leverages additional

state in the OpenFlow pipeline. Packet labels carry reroute and connectivity information. Braun et al. [1] propose loop detection for LFAs (LD-LFA) which increases the number of protected destinations but may erroneously drop packets. The authors of [37] use labels in the packet header that carry failure information to trigger rerouting in other nodes. Cevher et al. [38] implement MRCs in OpenFlow. The authors of [39] implement multi-topology routing which uses virtual topologies to provide redundancies in routing tables. If a failure is detected, packet forwarding is switched to a topology which is not affected by the failure. BOND [40] optimizes memory management for backup rules and leverages global hash tables to accelerate failure recovery.

3) P4-based FRR: P4 does not provide native FRR capabilities. Therefore, the hardest challenge is to provide the data plane devices with information about which neighbors are reachable, i.e., which port is up or down.

Sedar et al. [41] propose to use registers to store information about which egress port is up or down. Depending on the port status registers, primary or backup forwarding actions are triggered. However, the authors depend on a local agent to populate the registers. Shared Queue Ring (SQR) [42] caches recent traffic in a delayed queue. If a link failure is detected, the cached traffic is sent over alternative paths. Lindner et al. [43] implement 1+1 protection in P4 which replicates traffic, includes sequence numbers, and sends it over disjoint paths. The joint head end of those paths deduplicates the traffic. Hirata et al. [44] implement a FRR scheme in P4 which is similar to MRCs. Multiple routing topologies with disjoint paths are deployed. A field in the packet header identifies the topology which should be used for forwarding. D2R [45] is a resilience mechanism which works entirely in the data plane. When a failure is detected, the data plane itself, i.e., the failure-detecting switch, recomputes a new path to the destination. A primitive for reconfigurable fast reroute (PURR) [46] stores additional egress ports for each destination. During packet processing, the first working egress port is selected for forwarding.

III. LFAs: State of the Art

We review LFAs and remote LFAs (rLFAs) and give an overview of previous work regarding loop detection for LFAs. Finally, we explain topology-independent LFAs (TI-LFAs).

A. LFAs and rLFAs

In this subsection we introduce the concept of LFAs and rLFAs. Then, we discuss three important properties of LFAs. First, we differentiate protection levels for LFAs, i.e., link protection and node protection. Second, we explain the influence of links cost on LFA-based protection. Third, we point out that LFAs may generate loops under some conditions.

1) Concept: LFAs [20] have been proposed in the context of FRR for IP networks to quickly protect traffic against the failure of links and nodes while primary forwarding entries are recomputed.

A point of local repair (PLR) denotes a node that detects an unreachable next-hop and reroutes affected traffic to some other neighbor. However, some neighbors would send the traffic back to the PLR, which creates a loop. The other neighbors can forward the traffic without creating a loop and are called loop-free alternates (LFAs). They are used by a PLR to reroute traffic in case of a failure.

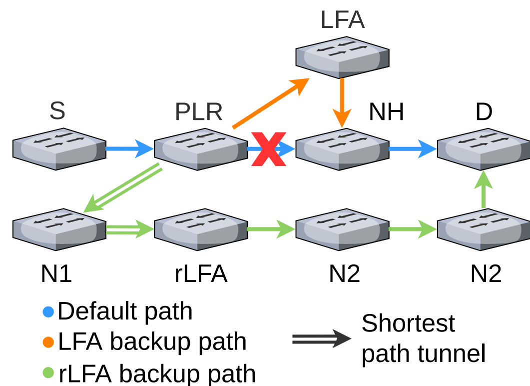


Figure 1: In case of a failure, a PLR may reroute a packet to an LFA or tunnel it via a shortest path to a rLFA. The (r)LFA then forwards the packet via a shortest path to its destination.

LFAs are illustrated in Figure 1. Traffic is forwarded on shortest paths. A packet is sent from sender *S* to destination *D*. The default path is via *PLR* and *NH*. When *PLR* cannot reach its next-hop *NH* due to a link failure, it cannot reroute the packet via neighbors *S* or *N1* as they forward traffic towards *D* to *PLR*, which creates a loop. However, *PLR* may reroute the packet via *LFA* which can forward the packet to *D*. Thus, the node *LFA* represents an LFA for *PLR* with respect to destination *D*.

We now assume that *NH* fails so that *LFA* has no working path towards *D*. If *PLR* reroutes the packet to *LFA*, *LFA* may use *PLR* as an LFA and return the packet. Thus, a loop occurs.

Remote LFAs (rLFAs) [23]–[25] have been introduced to protect more destinations than LFAs by sending packets through shortest path tunnels to remote nodes. In our example, the node *rLFA* is an rLFA for *PLR* with respect to destination *D*. If *NH* fails, *PLR* may tunnel the packet to *rLFA* which decapsulates the packet and sends it to *D* via a shortest path.

2) Protection Level: We already observed that some (r)LFAs protect only against link failures, others protect also against node failures. The first are classified as link-protecting (LP), the second as node protecting (NP). A link-protecting LFA (LP-LFA) forwards traffic to a destination via a path that avoids a PLR's failed link. A node-protecting LFA (NP-LFA) forwards traffic to a destination via a path that avoids a PLR's failed next-hop. Thus, NP-LFAs are also LP-LFAs, but not vice-versa. Therefore, a PLR can protect more destinations with LP-LFAs than with NP-LFAs. For some destinations, there may be no LP-LFA or NP-LFA at all. Then, rLFAs may help.

3) Influence of Link Cost: Networks are configured without link costs, i.e., unit link cost networks, or with link costs, i.e., non-unit link cost networks, e.g., for traffic-engineering. (r)LFAs have different protection properties in unit link cost networks than in non-unit link cost networks. The authors of

[1], [2] showed that for some destinations there is no LP-LFA or NP-LFA in both unit-link cost networks and non-unit link cost networks. Then, some destinations may be protected with rLFAs. Csikor et al. [25] proved that there is always an LP-rLFA for any destination in unit link cost networks. However, in [2] we showed that this is not the case in non-unit link cost networks. Furthermore, we showed that in both unit link cost networks and non-unit link cost networks there is not always an NP-rLFA for a destination [2] although there are more NP-rLFAs in unit link cost networks. Thus, in general, more destinations can be protected in unit link cost networks with (r)LFAs than in non-unit link cost networks.

4) LFA-Generated Loops: Forwarding loops in networks are problematic for two reasons. First, the traffic cannot reach its destination. Second, looping traffic consumes bandwidth, which may lead to packet loss for other traffic. However, looping traffic does not loop forever because the TTL field in the IP header limits the number of forwarding hops. As TTL=64 is a typical value, looping traffic can easily waste the 30-fold of the capacity it would normally occupy on a link. Therefore, routing loops are detrimental and should be avoided.

Depending on their protection level (r)LFAs may cause rerouting loops in specific failure scenarios. We distinguish and order four failure scenarios: single link failure (SLF) $<$ single node failure (SNF) $<$ double link failure (DLF) $<$ single link and single node failure (SLF+SNF).

LP-(r)LFAs do not cause rerouting loops for SLF but they may cause loops in other scenarios. NP-(r)LFAs prevent loops for both SLF and SNF [2], but fewer destinations can be protected by them. In case of multiple failures, even NP-(r)LFAs may generate loops. Some LP- or NP-(r)LFAs have the "downstream" property [12] and they avoid loops in case of multiple failures. However, only a few LFAs have that property so that only a few destinations can be protected by them. We do not consider them any further in this study.

B. Loop Detection for LFAs

The authors of [1] propose loop detection based on bit strings. They use it in combination with LFAs to protect more destinations by LFAs without suffering from loops. In addition, they suggest to protect destinations with LFAs with the highest possible protection level to maximize the coverage against link and node failures. They call this approach LD-LFA.

1) Loop Detection Based on Bit Strings: The loop detection in [1] requires a bit string in the packet header to indicate nodes that have rerouted the packet before. Each node in the network is associated with a bit position. If a packet is rerouted, the node activates it bit in the packet's header. If a node receives a packet with its corresponding bit activated, the packet is dropped.

The authors suggest an implementation in OpenFlow but do not deliver a prototype. An advantage of this approach is that a packet can be rerouted by multiple nodes. A disadvantage is the missing scalability. Bit strings in packet headers should be small. In OpenFlow, MPLS labels may be reused for that purpose, but they are only 4 bytes long which is not enough to number all nodes of a large network. Therefore, multiple nodes may be associated with the same bit. If one of these nodes reroutes a packet, the packet is dropped if it is received by another of those nodes. This causes erroneous drops for rerouted packets.

2) LFA Selection: For some PLRs there are several LFAs available for a specific destination. The authors of [1] suggested to prefer NP-LFAs over LP-LFAs in such a case. They showed for various network topologies that significantly fewer destinations can be protected by NP-LFAs than by LP-LFAs. Therefore, they suggested to protect the remaining destinations with LP-LFAs if possible. In addition, they proposed to utilize loop detection based on bit strings to avoid rerouting loops caused by LP-LFAs. They did not consider rLFAs.

C. Topology-Independent LFAs

In this subsection we explain topology-independent LFAs (TI-LFAs) [3]. First, we review segment routing (SR) [28]. Then, we describe TI-LFAs.

1) Segment Routing: IP networks leverage destination-based forwarding to deliver packets. That is, a packet carries the IP address of the destination in its header which is used by network devices to determine the appropriate next-hop according to entries in a forwarding table. In contrast, with SR the packet source determines the processing of a packet. To that end, SR leverages forwarding instructions in the packet header. The packet source constructs a set of header segments that are added to the packet. Each header segment corresponds to a specific action. Nodes process a packet according to the segments in its header. To that end, network devices maintain a certain number of forwarding entries to map a header segment to a specific action.

Currently, there are two major technologies that implement SR. SRv6 [47] is based on IPv6 and its extension header. Each IPv6 address in the extension header corresponds to one header segment. SR-MPLS ([48]) leverages stacked MPLS labels, i.e., the header stack, where each MPLS label is a header segment. To facilitate readability we only use the terminology of SR-MPLS, i.e., header stack and label, in the following.

Header segments may instruct nodes to perform arbitrary actions, e.g., forwarding a packet, pushing or removing other header segments, etc. In the following we focus on two specific types of header segments. The first type are header segments for global forwarding. We refer to such header segments with the term "global labels". Global labels instruct the nodes to forward a packet according to shortest paths towards a specific destination. As a result, a global label is similar to destination-based forwarding in IP networks. At the destination the global label is removed and the node processes the next header segment. When global labels are used for all destinations, every nodes requires $n-1$ forwarding entries where n is the number of destinations in the network. The second type are header segments for local forwarding. We refer to that kind of header segments with the term "local labels". Local labels instruct nodes to forward a packet over a specific link towards a next-hop. Before a node forwards a packet to the NH, it removes its local label from the header stack. When local

labels are used for all nodes, every node requires d forwarding entries where d is the number of neighbors of that node.

A source may construct a header stack that contains both global labels and local labels. As a result, forwarding differs depending on which type of label is on top of the header stack. On some subpaths the packet is forwarded according to a global label and on some subpaths the packet is forwarded according to a local label.

2) Concept of TI-LFAs: TI-LFAs leverage SR to forward packets on explicit paths around a failure. That is, TI-LFAs are not restricted to shortest paths because they construct a header stack with explicit forwarding instructions so that the packet avoids the failure. As a result, TI-LFAs with LP protect against any single link failure independently of link costs, and TI-LFAs with NP protect against any single node failure independently of link costs. However, multiple header segments may be necessary which increases the size of the header stack and thereby the overhead in terms of additional packet headers. The authors of TI-LFAs state that "in an MPLS world, this may create a long stack of labels to be pushed that some hardware may not be able to push." ([3], 2021, p. 6).

The size of a specific header stack depends on how the explicit backup path is implemented. The straightforward approach is to use one explicit forwarding instruction for every hop, i.e., local labels. However, this requires one header segment for each hop which causes large header stacks. The size of the header stack can be reduced if subpaths of the explicit path are implemented with already existing global labels. That is, one global label replaces multiple local labels. This is possible when working shortest paths are subpaths of the explicit path. However, this may not be possible for all subpaths because sometimes no working shortest subpath is available due to the failure.

The authors of [3] do not specify how the header stack to implement explicit paths is built. In particular, this is an optimization that highly depends on the failure scenario, topology, link costs, and path selection. Therefore, we see research potential for the optimization of the TI-LFA header stack. This, however, is out of scope of this document. In the following we assume that TI-LFAs implement explicit paths only with local labels.

IV. Robust LFA Protection for Software-Defined Networks (RoLPS)

LFAs originated from IP networks. They are attractive for SDN because they entail only little overhead in terms of additional forwarding state. However, they have three major shortcomings. They have been designed only for shortest-path routing based on link costs, they cannot protect all destinations, and they may cause loops under some conditions.

In the following we explain how LFAs can be applied in SDN which allows for general destination-based forwarding. We present explicit LFAs so that all destinations can be protected in case of a failure, provided they can be physically reached by a working path. We describe an advanced loop detection method to detect and stop loops and prevent erroneous packet drop after up to n reroute actions. Finally, we propose how to utilize these components and consider different protection variants.

A. Applicability of LFAs for SDN

In the context of IP networks, equations considering link costs are used to classify neighboring nodes into non-LFAs, LP-LFAs, and NP-LFAs with regard to some destination [12]. Forwarding in SDN does not need to follow shortest path routing based on link costs, but general destination-based forwarding may be applied. Therefore, we briefly explain how (r)LFAs can be used in that context. Essentially, we need to classify neighboring nodes into no-LFAs, LP-LFAs, and NP-LFAs. A PLR's neighboring node is

- no LFA if its standard forwarding procedure forwards the traffic to the destination via a path containing the PLR.
- an LP-LFA if its standard forwarding procedure forwards the traffic to the destination via a path that does not contain the link from PLR to its next-hop towards the destination.
- an NP-LFAs if its standard forwarding behavior forwards the traffic to the destination via a path that does not contain the PLR's next-hop towards the destination.

This definition can be applied to normal LFAs, rLFAs, and to eLFAs that are presented later in this section.

Path computation is not a focus of this paper. To limit the parameter space for ease of understanding, we consider in the evaluation in Section V link-cost-based forwarding which is a special case of the more general destination-based forwarding.

B. Explicit LFAs

We first give an example where (r)LFAs cannot protect a destination. Such destinations can be protected by explicit LFAs (eLFAs) which are based on explicit tunnels. However, explicit tunnels require additional forwarding entries. In [2] we suggested to implement explicit tunnels with explicit point-to-point rerouting tunnels. In this paper, we propose explicit multipoint-to-point rerouting tunnels as an alternative which requires significantly less additional forwarding entries. Finally, we explain the relation between eLFAs and TI-LFAs.

1) Protection through Explicit Tunnels: The network in Figure 2 forwards traffic on shortest paths based on costs that are annotated on the links. *PLR* sends a packet to *D* but the primary next-hop is unreachable. Although there is a physical path via *N1* and *eLFA*, there is no (r)LFA available. *N1* is not an LFA because it sends traffic to *D* via *PLR*. *eLFA* cannot serve as rLFA because the shortest path from *PLR* to *eLFA* traverses *D*. The problem can be solved by setting up an explicit tunnel via *N1* to *eLFA* a priori. If *D* is no longer reachable, *PLR* can send the packet over that explicit tunnel, and from *eLFA* the packet reaches *D* via a shortest path. Thus, *eLFA* is an eLFA for *PLR* with regard to *D*.

2) Explicit Point-to-Point Rerouting Tunnels: Now we explain the concept of explicit point-to-point tunnels which we introduced in [2]. In Subsection VI-C3 we describe technical details about the implementation of explicit tunnels in general with P4.

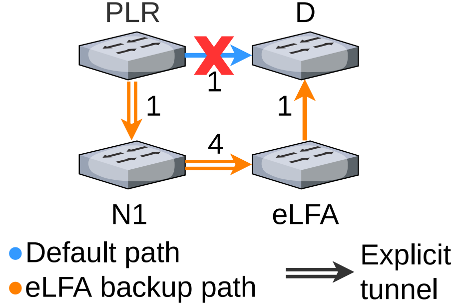


Figure 2: In case of a failure, a PLR may reroute a packet to an eLFA via an explicit tunnel which then forwards the packet via a shortest path to its destination. In contrast to rLFAs, the PLR cannot reach the eLFA via a shortest path.

Explicit point-to-point rerouting tunnels do not follow standard paths. Therefore, they are configured with a unique identifier, e.g., a unique number or IP address, in advance. When a PLR reroutes a packet through an explicit point-to-point rerouting tunnel, it adds the identifier of that tunnel to the packet. Nodes use the identifier to forward the packet along the explicit path. To that end, the nodes along an explicit path need additional forwarding entries for the identifier of that tunnel. Additional forwarding entries for FRR purposes are undesired overhead for the data plane as they limit its scalability.

3) Explicit Multipoint-to-Point Rerouting Tunnels: The overhead of additional forwarding entries from explicit tunnels can be reduced by using explicit multipoint-to-point tunnels. That is, the explicit tunnels from multiple PLRs towards the same endpoint, i.e., an eLFA, build a destination tree where the PLRs are the sources and the eLFA is the sink. Such an explicit multipoint-to-point rerouting tunnel corresponds to a specific eLFA and is identified by a single unique identifier. When a PLR reroutes a packet towards a specific eLFA, it adds the identifier of the corresponding multipoint-to-point rerouting tunnel to the packet. As a result, overlapping subpaths of explicit tunnels towards the same eLFA require only a single additional forwarding entry in nodes along that subpath. Therefore, multipoint-to-point rerouting tunnels are prefered over point-to-point rerouting tunnels. We evaluate the effect of multipoint-to-point rerouting tunnels in comparison to point-to-point rerouting tunnels in Section V-C.

4) Relation to TI-LFAs: Explicit tunnels can be implemented in different ways. We suggest eLFAs which implement explicit tunnels with a single tunnel header and additional forwarding entries in forwarding devices. Alternatively, TI-LFAs leverage a header stack with explicit forwarding instructions based on already available forwarding entries. Section III-C contains details about the construction of the TI-LFA header stack. Either way creates overhead to implement explicit tunnels. In Section V-C we evaluate the number of additional forwarding entries that are required by eLFAs. In Section V-D we quantify the size of the packet header stack when TI-LFAs are used.

C. Advanced Loop Detection

The loop detection method in [1] suffered from scalability problems. Therefore, we propose that packets are dropped if they are rerouted more than n times. This requires only a counter in the packet header which is increased with each reroute action. When the counter reaches the limit, the packet is dropped. We denote this advanced loop detection (ALD). Generally, ALD can be configured to support an arbitrary number of redirects. However, a large number can be counter-productive as packets are dropped later in case of loops and consume more bandwidth. In our context, we allow a packet to be rerouted twice so that double failures can be survived.

1) Implementation in OpenFlow: Due to technical restrictions of OpenFlow, conditions can be checked only at the beginning of the forwarding pipeline. However, at that stage, there is no knowledge about the packet's next hop and failed interfaces. Fortunately, it is possible to increase the reroute counter while rerouting. Thus, only the next-hop of a rerouted packet can determine whether the packet's reroute counter exceeds the limit and then the packet is dropped. This wastes bandwidth on the last link over which the packet was rerouted.

We provided a more detailed sketch of an OpenFlow-based implementation of ALD in [2]. That particular proposal was still based on bit strings. However, it avoids erroneous packet drops after a single reroute in contrast to the solution in [1].

2) Implementation in P4: P4 offers more implementation flexibility. Therefore, it is possible to check whether a packet is rerouted and whether its rerouting counter exceeds the limit before the packet is forwarded to the egress port. As a consequence, packets are dropped before transmission, which does not waste bandwidth. More details about the P4-based implementation of ALD are given in Section VI-D.

D. RoLPS Protection Variants

With SDN a controller configures flow entries on data plane devices. Alternative paths can be configured so that the device can switch over to a secondary next-hop if the first hop becomes unreachable. The secondary next-hop is also configured by the controller. In this section we present a ranking scheme for LFAs to choose the best one as a secondary next-hop. We further define protection variants and propose a corresponding nomenclature.

1) LFA Ranking: A controller can classify neighboring and remote nodes of a potential PLR into LFAs, rLFAs, and eLFAs, and as LP or NP for a specific destination. These LFAs can be ranked according to their protection level, i.e., NP is better than LP. Recall that NP-LFAs are also LP-LFAs, but not

Rank	LFA Type
0	NP-LFA
1	NP-rLFA
2	NP-eLFA
3	LP-LFA
4	LP-rLFA
5	LP-eLFA

Table 1: Ranking of LFA types according to protection level and complexity. Preference is given to LFAs with lower rank number.

Mechanism	C-LFA (nLD-LP-LFA)	C-rLFA (nLD-LP-rLFA)	LD-LFA (ALD-NP-LFA)	ALD-NP-rLFA	ALD-LP-eLFA	ALD-NP-eLFA
Loop detection			•	•	•	•
Protection against all SLF		o		o	•	•
Protection against all SNF						•
Additional forwarding entries					•	•

Table 2: Properties of protection variants.
Legend: o = only for unit link costs; • = independent of link costs.

vice-versa. They can also be ranked according to complexity. Normal LFAs are simplest as they do not require tunneling. eLFAs are most complex as they entail additional forwarding entries for explicit tunnels.

With SDN, it is important to have an alternative next-hop in case the primary next-hop is unreachable as it may take too long until the forwarding is fixed by the controller. Therefore, we rank LFAs first according to their protection level and then according to their complexity. This yields the ranking given in Table 1. The ranking is used to select the best available LFA during computation.

2) Protection Variants: We define several protection variants with respect to loop detection, LFA complexity, and protection level. The following naming scheme is used: {nLD, ALD}-{LP, NP}-{LFA, rLFA, eLFA}. Loop detection may be activated or not {ALD, nLD}. Either the LP property is sufficient or NP is desired {LP, NP}. Only normal LFAs may be allowed, normal and rLFAs may be allowed, or normal, remote, and explicit LFAs are supported {LFA, rLFA, eLFA}.

eLFAs are preferably implemented with explicit multipoint-to-point rerouting tunnels (see Section IV-B3). However, for comparison we sometimes refer to eLFAs with point-to-point tunnels. To that end, we add the suffix "-p2p" to the protection variant. We omit a suffix for eLFAs with multipoint-to-point rerouting tunnels because this is the preferable way That is, *-*-eLFA refers to protection variant with eLFAs with multipoint-to-point rerouting tunnels and *-*-eLFA-p2p refers to protection variants with eLFAs with point-to-point rerouting tunnels.

If a protection variant requires the NP property, the LFA selection process starts with the search for an LFA of rank 0. If the search is successful, this LFA is configured as secondary next-hop for a specific destination, and the algorithm stops. Otherwise the search continues with the next higher rank number. This possibly continues up to rank 5. That means, NP-(e/r)LFAs are preferentially utilized, but LP-(e/r)LFAs may be used if the destination cannot be protected otherwise. This is needed, e.g., if the protected next-hop is the destination. If no LFA has been found for the last rank, there is no physical connection between PLR and destination.

If a protection variant requires only the LP property, the LFA selection process starts with the search for an LFA of rank 3. The algorithm also stops if no LFAs has been found for the last rank. In that case there is no physical path between PLR and destination. Note that LFAs of rank 3 may also be NP as every NP-LFA also fulfills the LP property. LP-LFAs are just not preferred over NP-LFAs when the protection variant requires only the LP property.

Protection variants requiring the NP property may still suffer from loops since some destinations can be protected only with LP-(e/r)LFAs. For example they occur when the destination of a flow fails. nLD-LP-LFA and nLD-LP-rLFA leverage only the classic LP-LFAs [20] and LP-rLFAs [23]. They are widely used in IP networks and we denote them as the classic LFA and rLFA variants (C-LFA, C-rLFA). ALD-NP-LFA[1] has been investigated as a preferred protection variant in [1] under the name LD-LFA.

Table 2 summarizes the most important protection variants investigated in our study. It summarizes properties regarding protection level and complexity. ALD-mechanisms prevent loops in any failure scenario. *-*-rLFA protect against all protectable SLF in networks with unit link costs. *-*-eLFA methods achieve that protection level even in networks with non-unit link costs. *-NP-eLFA protects even against all protectable SNF in networks with either unit or non-unit link costs.

V. Simulative Performance Evaluation of LFA-Based Protection

In this section we analyze the efficiency of LFA-based FRR mechanisms. First, we describe the methodology. The performance metrics of interest are protection coverage, required amount of additional forwarding entries, required amount of header segments for TI-LFAs, and path lengths. We compare them for RoLPS protection variants and other well-known FRR mechanisms. Finally, we discuss the presented results.

A. Methodology

We explain the methodology for the simulation-based evaluation. We describe the general approach, and discuss the topology data set and link costs used in the evaluation.

1) General Approach: We take a network topology including link costs and a RoLPS protection variant as input parameters. Then we compute LFAs according to Section IV-D. We evaluate different protection variants against various sets of failure scenarios, i.e., $\mathcal{S} \in \{\mathrm{SLF}, \mathrm{SNF}, \mathrm{DLF}, \mathrm{SLF{+}SNF}\}$ (see Section III-A4). To that end, we consider all source-destination pairs $f \in \mathcal{F}$ in the network and analyze how their traffic is forwarded in a specific failure scenario $s \in \mathcal{S}$.

Although RoLPS works for general destination-based forwarding (see Section IV-A), we limit the evaluation to shortest paths routing based on link costs to reduce the parameter space.

[1]Approximation of LD-LFAs with better loop detection.

2) Network Topologies: We evaluate 205 wide area, commercial, research, and academic networks from the Internet topology zoo [49] and three typical data center topologies (fat-tree, DCell, BCube) which were studied in [1]. For each topology we calculate both average values and maximum values for the considered metrics. We explain these metrics in Sections V-B1, V-C1, and V-E1. We visualize the results in bar diagrams or complementary cumulative distribution functions (CCDFs).

3) Link Costs: In Subsection III-A3 we showed that link costs have a significant impact on the protection properties of LFAs. To account for that fact, we perform evaluations on then networks with both unit link cost and non-unit link cost. However, the topology zoo does not include link costs for all networks. Therefore, we calculate link costs on all networks as proposed in [50]. For each link we derive the specific load based on a homogeneous traffic matrix, shortest paths, and unit link costs. The link cost of each link is the inverse of its load multiplied by the largest link load in the network so that the smallest link cost is 1. Over all topologies this leads to an average link cost of 6.8 and a coefficient of variation of link costs of 1. Thus, the generated link costs differ substantially.

B. Protection Coverage

In this subsection we evaluate and compare the coverage of RoLPS protection variants. First, we explain the metric. Then, we briefly describe the evaluated protection mechanisms. Finally, we discuss results for networks with unit link costs and with non-unit link costs.

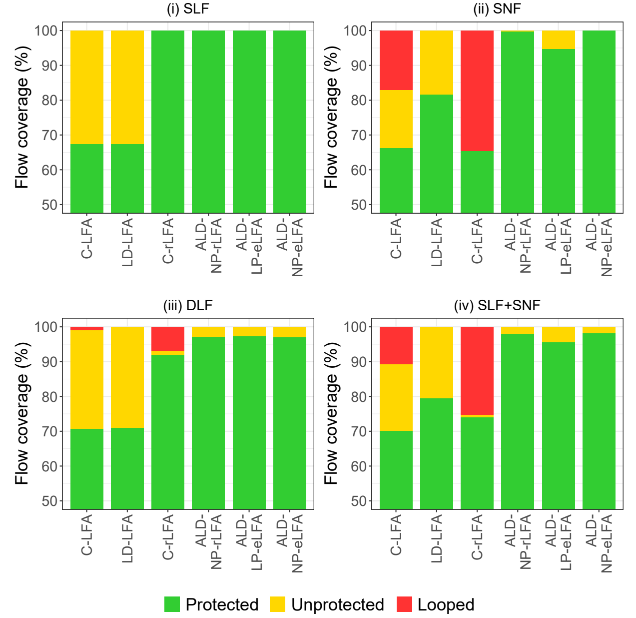
1) Metric: We introduce the three terms 'protected', 'unprotected', and 'looped' to refer to the quality of protection which is provided by a FRR mechanism for a flow in a specific scenario that consists of topology, failure scenario, and link costs. A flow is considered protected in two cases. First, if the packet is still successfully delivered at the destination although the path from source to destination was interrupted by a failure. Second, if a packet is dropped to prevent a loop because the destination is not reachable anymore. A flow is unprotected if the packet is dropped although the destination is still reachable. Finally, a flow is denoted as looped if a microloop was caused by local rerouting. We report the average fraction of protected, unprotected, and looped flows over all 208 topologies (see Section V-A2) in bar diagrams. The term coverage refers to the fraction of protected flows in a scenario.

2) Evaluated Protection Variants: We consider the classic protection variants C-LFA (nLD-LP-LFA) and C-rLFA (nLD-LP-rLFA) as well as the LD-LFA (ALD-NP-LFA) from [1]. We further study the new protection variants ALD-NP-rLFA and ALD-{LP,NP}-eLFA since they have stronger protection properties.

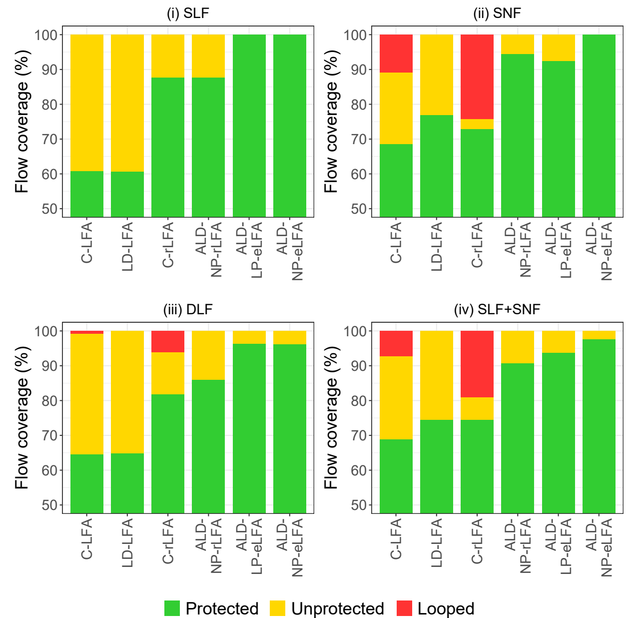
3) Coverage: In this section we present results for the number of protected destinations for different failure scenarios. First, we evaluate unit link cost networks. Then, we discuss non-unit link cost networks.

a) Networks with Unit Link Costs: Figure Figure 3(a) shows the coverage in percent for different sets of failure scenarios in networks with unit link costs. Subfigure 3(a) (i) shows that only C-LFA and LD-LFA cannot protect all destinations against SLF, i.e., their coverage is less than 100%. All other protection variants provide full coverage.

Subfigure 3(a) (ii) shows that SNFs cause many rerouting loops with C-LFA (17%) and C-rLFA (34%). This is mostly caused by failed destinations. As C-rLFA protect more destinations than C-LFA, they also cause more loops when the next-hop is the destination. Thus, loop detection is even more important when C-rLFA is used because more flows loop in case of node failures than with C-LFA. LD-LFA protects more



(a) Networks with unit link costs.



(b) Networks with non-unit link costs.

Figure 3: Coverage averaged over 208 topologies depending on protection method and set of failure scenarios.

traffic (81%) than C-(r)LFA in case of SNF as it preferentially uses NP-LFAs if available. Moreover, it prevents loops.

The new protection variants have significantly higher coverage. ALD-NP-rLFA protects around 99% of the destinations with SNF. This results from dropping packets that cannot be delivered anymore due to a failed destination; if they looped, the corresponding flow would count as looped. The coverage of ALD-LP-eLFA is slightly lower, i.e., 94%. This is because NP-(e/r)LFAs are not preferentially chosen for this protection variant so that there are more LFAs in use without the NP property. Finally, ALD-NP-eLFA protects all destinations for three reasons. First, it leverages rLFAs or eLFAs to provide protection for destinations that cannot be protected with LFAs. Second, it uses NP-(e/r)LFAs to protect against node failures and falls back to unsafe LP-(e/r)LFAs only when (e/r)LFAs with NP property are not available. Third, ALD detects and stops all loops that may be caused by LFAs with LP. This turns flows that cannot reach their destination into protected flows instead of looped flows.

Subfigure 3(a) (iii) shows the coverage against DLFs. No mechanism is able to protect all destinations. C-LFA and LD-LFA protect around 70% of the destinations. C-rLFA cover more flows (92%). However, protection variants without loop detection, i.e., C-LFA and C-rLFA, lead to loops. All newly proposed protection variants achieve roughly the same coverage, i.e., 96%, and prevent loops.

Finally, Subfigure 3(a) (iv) shows results for SLF+SNF. They are similar to the results of DLFs, but the fraction of rerouting loops caused by both C-LFA and C-rLFA is significantly higher. This is due to node failures which cause significant rerouting loops for protection variants without loop detection.

b) Networks with Non-Unit Link Costs: Figure Figure 3(b) shows the coverage for different sets of failure scenarios in networks with non-unit link costs. Subfigure 3(b) (i) shows the coverage against SLF. Both C-LFA and LD-LFA protect only around 60% of the destinations. In networks with non-unit link costs, C-rLFA cannot protect all destinations anymore against SLF and achieve only a coverage of 88%. The same holds for ALD-NP-rLFA. Only the eLFA-based protection variants are able to protect all destinations against SLF.

Subfigure 3(b) (ii) shows the coverage against SNF. Both C-LFA and C-rLFA cause many rerouting loops. LD-LFA prevents loops but protects only 76% of the destinations. ALD-NP-rLFA and ALD-LP-eLFA protect a higher fraction of destinations, i.e., 94% and 93%, because they prevent loops of unsafe LFAs with LP, but they have no suitable backup path for some node failures. ALD-NP-eLFA protects all destinations against SNF even in networks with non-unit link costs as it prevents loops and leverages NP-(e/r)LFAs wherever possible.

Finally, Subfigure 3(b) (iii) and Subfigure 3(b) (iv) present the coverage for DLF and SLF+SNF. The results are similar to those from networks with unit link costs, but the coverage here is slightly lower.

C. Additional Forwarding Entries

We now evaluate the number of additional forwarding entries to implement explicit tunnels. First, we explain the metric. Then, we discuss the investigated FRR mechanisms. Finally, we present results for networks with unit link costs and non-unit link costs.

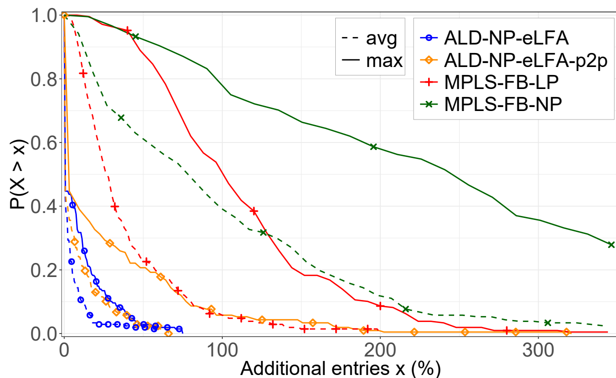
1) Metric: In a network with n nodes, each node maintains $n-1$ forwarding entries for destination-based forwarding. eLFAs require additional forwarding entries to implement explicit tunnels. In contrast, both LFAs and rLFAs are based on shortest paths, and therefore, do not need additional forwarding entries. We calculate the average and maximum amount of additional forwarding entries per node relative to $n-1$ for each network and present the results for all topologies in a CCDF.

2) FRR Mechanisms under Study: We compare the required amount of additional forwarding entries only for eLFA-based RoLPS protection variants as others do not require additional forwarding entries. To evaluate the efficiency of multipoint-to-point rerouting rerouting tunnels, we report results for ALD-{LP,NP}-eLFA and compare them to the corresponding mechanisms with point-to-point rerouting tunnels, i.e., ALD-{LP,NP}-eLFA-p2p. In addition, we present results for state-of-the-art MPLS-facility-backup (MPLS-FB-{LP,NP}) with LP and NP property.

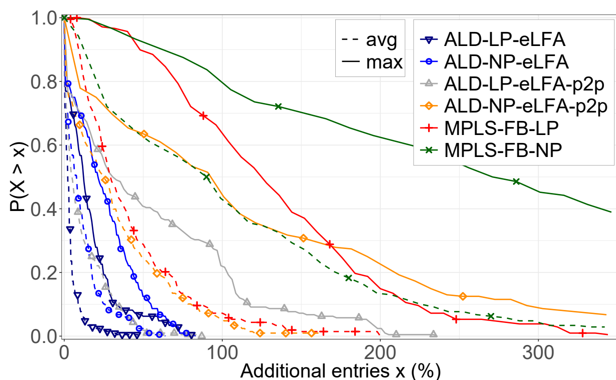
3) Results: We present results for the fraction of additional forwarding entries. First, we evaluate unit link cost networks. Then, we discuss non-unit link cost networks.

a) Networks with Unit Link Costs: Figure 4(a) shows a CCDF for the relative amount of additional forwarding entries for the considered FRR mechanisms in networks with non-unit link costs. First, we compare LP mechanisms. With MPLS-FB-LP, in 40% of the networks at least one node requires 120% or more additional entries (max-curve). However, on average in only 6% of the networks more than 100% additional entries are needed (avg-curve). The curves for ALD-LP-eLFA and ALD-LP-eLFA-p2p are omitted because those protection variants do not induce any additional forwarding entries. This is because (r)LFAs alone protect all destinations against all SLF in networks with unit link costs. Therefore, explicit LFAs are not needed and no additional forwarding entries are required.

Now, we compare NP mechanisms. MPLS-FB-NP requires most additional entries by far. 62% of the topologies have at least one node that requires 200% or more additional entries. And in 40% of the topologies 100% or more additional entries are required on average. Protection mechanisms with eLFAs, i.e., ALD-NP-eLFA and ALD-NP-eLFA-p2p, require less forwarding entries because they protect most of the destinations by NP-rLFAs and only the few remaining destinations are protected by eLFAs which induce forwarding state in the network. When ALD-NP-eLFA-p2p is used, only 20% of topologies have a node that requires 50% or more additional entries. However, some topologies contain at least one node that requires 200% or more additional entries. On average, no topology requires more than 65% or more additional entries. ALD-NP-eLFA is even more efficient because it leverages multipoint-to-point rerouting tunnels to reduce the number



(a) Networks with unit link costs. ALD-LP-eLFA does not induce any additional entries and is omitted in the figure.



(b) Networks with non-unit link costs.

Figure 4: CCDFs for fraction of additional forwarding entries.

of additional forwarding entries even further. There is no topology with a node that requires more than 70% of additional entries. 90% of the networks require only 15% or less additional entries on average.

b) Networks with Non-Unit Link Costs: Figure 4(b) shows a CCDF for the relative amount of additional forwarding entries for the considered FRR mechanisms in networks with non-unit link costs. Again, we compare LP mechanisms first. MPLS-FB-LP requires lots of additional entries. Around 55% of the topologies have at least one node that requires 120% or more additional entries (max-curve). However, in only 8% of the networks more than 100% additional entries are needed on average (avg-curve). eLFA-based protection mechanisms, i.e., ALD-LP-eLFA and ALD-LP-eLFA-p2p, are more efficient. When ALD-LP-eLFA-p2p is used, 22% of networks contain at least one node that requires 100% or more additional entries. On average, in 20% of networks nodes require 25% or more additional entries. ALD-LP-eLFA reduces the number of additional forwarding entries by leveraging multipoin-to-point tunnels. There is no topology with a node that requires more than 80% of additional entries and in 95% of the networks less than 15% additional entries are needed on average.

Now we compare NP mechanisms. MPLS-FB-NP requires most additional entries by far. 75% of networks have at least one node that requires 120% or more additional entries, 40% even more than 340%. In around 44% of the networks, 100% or more entries are required on average, and in 8% of the networks even 250% or more additional entries are required. ALD-NP-eLFA-p2p requires less entries. Only 45% of networks contain a node that requires 100% or more additional entries, but 20% of networks require even more than 210%. On average, 22% of networks require 50% or more additional entries. ALD-NP-eLFA is even more efficient. No network contains a node that requires more than 80% additional entries. In 90% of the networks, less than 30% additional entries are required on average.

Thus, in networks with non-unit link costs, somewhat more additional entries are needed but ALD-{LP,NP}-eLFA still require significantly less entries than MPLS-FB-{LP,NP} and ALD-{LP,NP}-eLFA-p2p.

D. Size of Header Stacks for TI-LFAs

In this section we evaluate the number of required segments to implement explicit paths with TI-LFAs using local labels. First, we explain the metric, and the studied mechanisms. Then, we present the results.

1) Metric: We count the number of header segments that are added to the packet by the respective mechanism for FRR purposes. That is, we do not count the header segment that identifies the original destination of the packet. For each network we record both the average and maximum number of additional header segments added to a packet and present the results as a CCDF.

2) Reroute Mechanisms under Study: We evaluate TI-LFAs that use only local labels (see Section III-C) because they implement explicit tunnels with multiple header segments. However, we leverage TI-LFAs only when there are no LFAs or rLFAs to protect a destination to avoid unnecessary additional header segments.

rLFAs, and eLFAs also leverage tunnels. However, both require only one additional header segment for tunneling which is why we omit those curves in the figure to facilitate readability. LFAs do not require additional header segments.

3) Results: Figure 5 shows the results for networks with non-unit link costs.

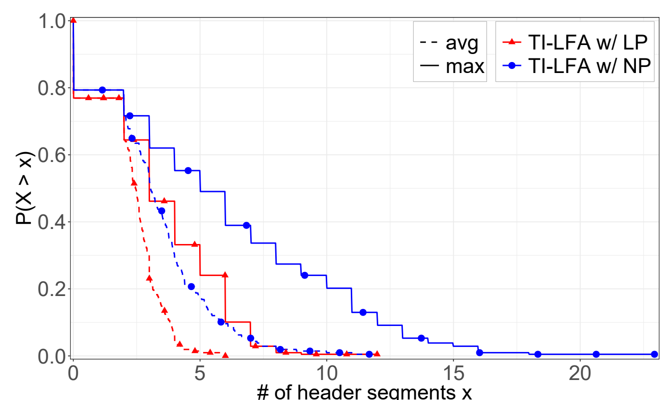


Figure 5: CCDF for number of additional header segments.

First, we discuss TI-LFAs with LP. In most networks, i.e., roughly 80%, TI-LFAs with LP require at least two additional header segments. In 20% of networks TI-LFAs with LP require on average 3 or more additional header segments. However, 23% of networks have at least one TI-LFA with LP that requires 6 or more additional header segments.

Now, we discuss TI-LFAs with NP. In general, TI-LFAs with NP require more additional header segments than TI-LFAs with LP. In 19% of networks TI-LFAs with NP require on average 5 or more additional header segments. 40% of networks even contain at least one TI-LFA with NP that requires 6 or more additional header segments.

In Section III-C2 we mentioned that the size of the TI-LFA header stack may be reduced. This, however, requires optimization which is a promising approach and an interesting research issue, but it is out of the scope of this document.

We omit a figure for results for networks with unit-link costs because of two reasons. First, TI-LFAs with NP require slightly fewer header segments but the results show no further insights. Second, for LP all destinations can be protected with either LFAs or rLFAs, i.e., no TI-LFAs are used, which was to be expected.

E. Path Lengths

In this section we report results for path lengths. First, we explain the metric and evaluated FRR mechanisms, then, we present the results.

1) Metric: We measure the path lengths of all flows that are affected by SLF but were successfully delivered due to local rerouting. For each topology, we calculate the average and maximum path lengths and present the results for all topologies in a CCDF.

2) Reroute Mechanisms under Study: We choose path lengths for rerouting as a baseline which recomputes shortest paths after a failure. We compare these results to the ones for ALD-{LP,NP}-eLFA and MPLS-FB-{LP,NP}.

3) Results: Figure 6 shows a CCDF for average and maximum path lengths of successfully delivered flows with SLF in networks with unit link costs.

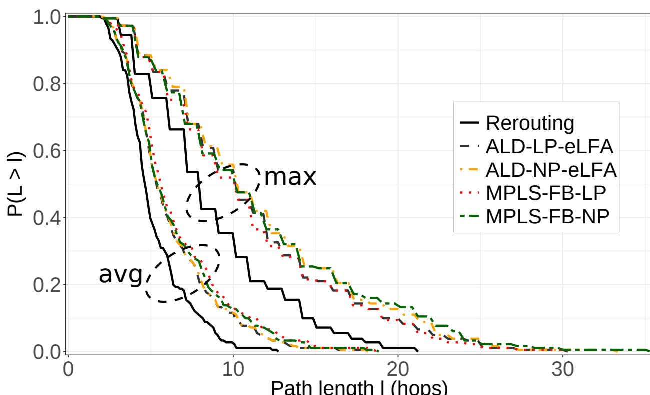


Figure 6: CCDF for path lengths of successfully delivered flows for SLF in networks with unit link costs.

We observe that rerouting leads in fact to the shortest maximum and average path lengths. All FRR mechanisms under study lead to longer maximum and average path lengths. The path lengths of the different FRR mechanisms does not differ.

The same analysis in networks with non-unit link costs leads to slightly longer paths but without any further insights. Therefore, we omit the corresponding figure.

F. Discussion

We investigated various RoLPS protection variants with regard to protection coverage, additional forwarding entries, and path lengths on a set of 208 topologies with both unit link costs and non-unit link costs, and compared them with MPLS-facility-backup.

The evaluations of protection coverage showed that C-LFA cannot protect many destinations in case of link failures. C-rLFAs can protect all destinations in case of SLF in networks with unit link costs. However, the usage of C-(r)LFA leads to many loops in case of node failures. The use of ALD avoids such loops. LD-LFA [1] prevents loops but cannot protect all destinations. ALD-NP-eLFA protects all destinations against SLF and SNF in networks with unit and non-unit link costs because it leverages eLFAs to complement (r)LFAs.

The explicit LFAs induce additional forwarding entries in the data plane, which is not desired. Therefore, we compared the additional forwarding entries for ALD-{LP,NP}-eLFA, ALD-{LP,NP}-eLFA-p2p, and MPLS-FB-{LP,NP}. ALD-{LP,NP}-eLFA require only very few additional entries compared to ALD-{LP,NP}-eLFA-p2p, and MPLS facility backup. Both MRCs [14] and IDAGs [18] always require 100% additional entries, and MRTs [15] need 200% more. Not-via addresses [11] need $100\% \cdot d$ more entries where d is the average node degree. Although TI-LFAs require at most d additional forwarding entries per node, they impose significant overhead in form of multiple additional header segments. ALD-{LP,NP}-eLFA add only one additional packet header for tunneling and our evaluation shows that they require less additional forwarding entries than other comparable FRR mechanisms. Therefore, ALD-{LP,NP}-eLFA can be considered very lightweight which makes them attractive for FRR in SDN.

All evaluated FRR mechanisms, i.e., ALD-{LP,NP}-eLFA and MPLS-FB-{LP,NP} extend backup paths by about the same, and backup paths are only slightly longer than the average and maximum length of recomputed shortest paths.

VI. Implementation of RoLPS in P4

We start with a short introduction of P4 and the implementation platform. Then we summarize important basics of P4 and describe the implementation of the RoLPS prototype.

A. Overview of P4 and the Implementation Target

P4 is a high-level programming language for protocol-independent packet processors [51]. P4 programs are mapped, i.e., compiled, to the programmable processing pipeline of

so-called targets, e.g., the software switch BMv2 [52] or the switching ASIC Tofino [53]. When a P4 program is successfully compiled for a target, it offers an API to let the control plane configure the device during runtime, e.g., to write forwarding entries.

In [2] we sketched how the predecessor of RoLPS could be implemented in OpenFlow. However, due to technical restrictions of OpenFlow the implementation concept required multiple workarounds which made it complex (see Section III-B1 and Section IV-C1). P4 offers significantly more flexibility than OpenFlow. It allows a flexible description of the data plane, in particular, the definition of arbitrary packet headers and packet parsers, and conditional application of programmable match+action tables (MATs). Therefore, implementation of novel features in P4 is easier than in OpenFlow.

In this paper we describe the implementation of RoLPS in P4. Our target is the P4-programmable high-performance switching ASIC Tofino [53] which is used in the Edgecore Wedge 100BF-32X [54] switch with 32 100 Gb/s ports. We made the source code for the RoLPS data plane and control plane publicly available[2].

B. P4 Pipeline

Figure 7 illustrates the abstract forwarding model of P4. A user-programmable parser extracts the information from the packet header and stores them in so-called header fields. They are carried with the packet through the processing pipeline, possibly with additional metadata which are similar to regular variables from other high-level programming languages. Metadata are packet-specific and discarded after the packet is sent to an egress port.

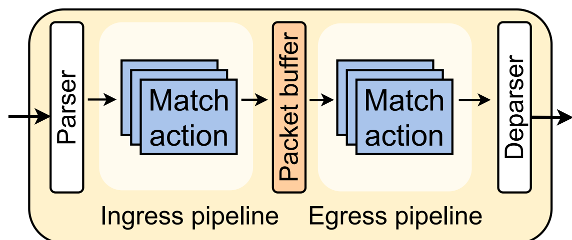


Figure 7: P4 abstract forwarding model according to [51].

The P4 abstract forwarding model is divided into two stages, the ingress and the egress pipeline, which are separated by a packet buffer. Match+action tables (MATs) allow for packet-specific processing. They have entries consisting of custom match fields and types that map header fields and metadata to actions, e.g., modifying header fields, and parameters.

P4 offers three match types: exact, longest-prefix match (LPM), and ternary. For an exact match the header field or metadata field must be exactly the same as the match field in the MAT, e.g., a specific IP address. LPM is well-known from standard IP forwarding. Ternary facilitates wildcard matches. P4 does not allow to match a packet multiple times on the same MAT to prevent processing loops.

[2]https://github.com/uni-tue-kn/p4-lfa

After the egress pipeline, the deparser writes the potentially modified header fields into the packet header and the packet is sent through the specified egress port.

However, P4 does not support FRR natively. Port status information cannot be accessed by the data plane by default. This makes the implementation of FRR in P4 a serious challenge.

C. Implementation of LFAs

First, we describe how the port status can be determined in P4. Afterwards, we describe the implementation of LFAs without tunnels followed by LFAs with tunnels, i.e., rLFAs and eLFAs, and ranking-based selection of LFA types.

1) Port Status Detection in P4: Executing backup actions, e.g., forwarding to an LFA, requires a reliable and timely detection when a port goes down. However, P4 does not support such a feature. In [55] we proposed a workaround for the Tofino platform which detects port-down events within 1 ms without controller interaction. We leverage this workaround to implement RoLPS-based protection and summarize it in the following.

Registers in P4 provide persistent storage, i.e., their content survives processed packets. The individual register fields can be accessed by an index. We leverage a register to store the current status of the egress ports by single bits (0: down, 1: up). Each register field stores the status of one port, i.e., one bit. The port ID serves as an index to access the corresponding register field. The challenge is updating the registers when the port status changes, which is platform-specific.

Port-down events are tracked as follows. Tofino has means outside the P4 programmable data plane to detect port-down events. We configured the Tofino such that it creates a 'port-down packet' in case of a port-down event. The packet contains the ID of the corresponding port and the packet is sent to a switch-intern port. We programmed the p4 pipeline such that the port status register for the respective port is set to zero upon reception of a port-down packet.

Port-up events are tracked differently. When the Tofino receives a packet over a specific port, it activates the status bit of that port in the register. To ensure that port-up events are detected sufficiently fast, we take advantage of topology packets that are regularly sent by the Tofino to all egress ports for neighbor detection. The frequency for topology packets can be configured to an appropriate value. While the detection of port-down events is time-critical, detection of port-up events is more relaxed because FRR mechanisms reroute affected traffic in the meantime via alternative ports.

2) Implementation of LFAs without Tunnels: As described in the previous section, the register fields provide information whether specific egress ports are up or down. However, the egress port of a packet is known only after matching the packet on a MAT. To mitigate this problem, we implemented FRR as shown in Figure 8. First, the packet is matched against a MAT that performs regular IPv4 routing, i.e., it determines the next-hop and thereby the egress port of a packet. Second, the ID of the selected egress port is used to access the register fields to retrieve the port status of that egress port. If the egress port is

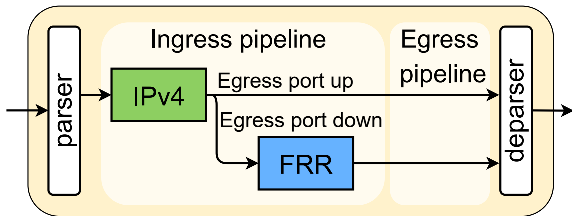


Figure 8: P4 implementation of FRR. A packet is matched against an IPv4 forwarding MAT to determine its egress port. If that port is down, the packet is matched against a FRR-MAT to determine its backup egress port.

up, the packet is forwarded. If the port is down, FRR actions are triggered, i.e., the packet is matched against a FRR-MAT using the IP destination address and the ID of the failed egress port. This selects a backup entry with a preinstalled LFA, i.e., backup egress port, for forwarding.

3) LFAs with Tunnels: LFAs with tunnels are implemented in a similar way as LFAs without tunnels. However, the backup actions in the FRR-MAT contain an encapsulation action which adds an additional IP header to the packet for tunneling to the remote node, i.e., the rLFA or eLFA.

If the remote node is an rLFA, the encapsulating IP header contains the IP address of that node. The packet is then forwarded on standard paths towards the rLFA.

If the remote node is an eLFA, the encapsulating IP header contains a unique IP address which identifies the explicit path towards the eLFA (see Section IV-B2). When the controller installs eLFAs in the network, it also sets up explicit tunnels towards the eLFAs. To that end, it calculates appropriate tunnel-specific forwarding entries and configures them on the forwarding devices along the explicit path. Thereby, the controller leverages explicit multipoint-to-point rerouting tunnels (see Section IV-B3) if possible to reduce the number of additional forwarding entries. That is, it configures only one additional forwarding entry on forwarding devices on overlapping subpaths of explicit paths towards the same eLFA.

4) Implementation of Ranking-Based Selection of LFA Types: The ranking-based selection of LFAs as described in Section IV-D is part of the control plane. The controller precomputes appropriate LFA types depending on the desired protection variant and installs corresponding egress ports and encapsulation actions in the FRR-MATs of the data plane devices.

D. Implementation of ALD

We implement ALD so that it allows two redirects, i.e., the packet is dropped when it has to be rerouted a third time. To that end, we define the ALD field as a 2-bit custom header field in the packet header. These bits track how often a packet has been rerouted. Packets initially carry the bit pattern ‘00’ in the ALD field. When a node reroutes a packet with bit pattern ‘00’, it replaces the bit pattern with ‘01’. When a node reroutes a packet with bit pattern ‘01’, it replaces the bit pattern with ‘10’. When a node cannot forward a packet with bit pattern ‘10’ due to a failed egress port, it drops the packet.

VII. Hardware-Based Performance Evaluation

In this section we conduct a performance evaluation of the RoLPS hardware prototype. It is based on the Tofino [53], a P4-programmable switch ASIC, which is used in the Edgecore Wedge 100BF-32X [54], a switch with 32 100 Gb/s ports. We present measurement results for throughput, restoration time, and loop detection.

A. Throughput

Every P4 program successfully compiled for the Tofino processes packets at a speed of 100 Gb/s. To verify that property for our prototype, we conducted the following experiment. We utilized an EXFO FTB-1 Pro traffic generator [56] which generates up to 100 Gb/s of traffic. We connected it to the Tofino which processes the traffic and sends it back to the traffic generator. This way we measure the traffic rate forwarded by Tofino. In fact, we obtained a throughput of 100 Gb/s for both failure-free forwarding and forwarding with activated FRR.

B. Restoration Time

The evaluation of restoration times is more complex. We describe the testbed, the measurement procedure and metric, as well as the experimental scenarios. Then, we present measurement results.

1) Testbed: Figure 9 shows the testbed for the performance evaluation. Center of the testbed is the above mentioned Tofino.

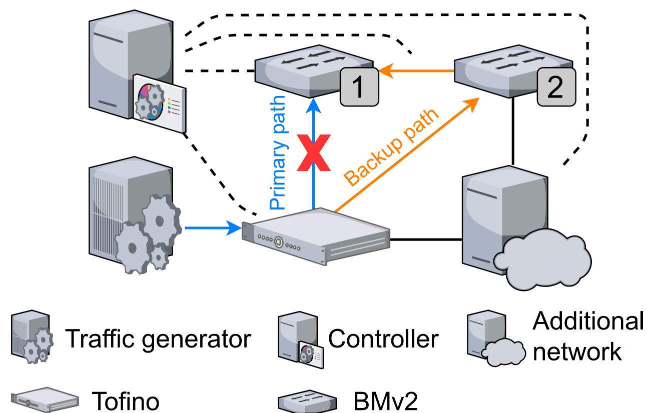


Figure 9: Topology for restoration time measurements. The additional network consists of five other BMv2s and 10 links.

It is connected to two BMv2 [52] P4 software switches. To perform evaluations for more realistic network sizes, we connected the Tofino to an additional network which consists of five BMv2s and 10 links. All BMv2s run on a server with an Intel Xeon Gold 6134 with 3.2 GHz and 12 cores, and 32 GB RAM. A controller is connected to the Tofino and all BMv2s. It configures them upon start, i.e., it discovers the topology, and computes and installs appropriate forwarding rules. It runs on the same server as the BMv2s. Furthermore, the above mentioned traffic generator is connected to the Tofino and serves as a traffic source in the experiment.

2) Measurement Procedure and Metric: The traffic generator sends traffic to the Tofino which forwards the packets on the primary path to the destination BMv2-1. BMv2-1 monitors the packet arrivals. Then, we deactivate the link from Tofino to BMv2-1 on the primary path to trigger a port-down event at the Tofino. We derive the restoration time for the FRR mechanism from a tcpdump log at BMv2-1. It is the duration of the interval within which BMv2-1 does not receive any packets.

In these experiments, the traffic generator sends only with 100 Mb/s instead of 100 Gb/s. This avoids overload on the BMv2s which can process packets only with around 900 Mb/s [57]. Avoiding overload is important only to obtain correct measurement results from BMv2-1. The restoration time on the Tofino is not affected by any overload.

3) Experiments: We perform two experiments to measure the restoration time without and with FRR.

a) Forwarding without FRR: For this experiment we disabled the FRR feature on Tofino. When the Tofino detects the failure, it notifies the controller. The controller then updates its topology, computes new forwarding entries, and installs them on the affected devices so that traffic can be forwarded again.

b) Forwarding with FRR: In this experiment the FRR feature is enabled. Thus, if BMv2-1 is no longer reachable, the Tofino forwards traffic destined to BMv2-1 to BMv2-2 which relays the traffic to BMv2-1.

4) Results: We performed the above described experiments 10 times. Figure 10 shows the average restoration time without and with FRR on the Tofino, including 95% confidence intervals.

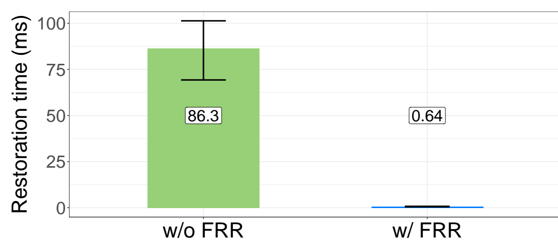


Figure 10: Restoration time on Tofino without and with FRR.

If FRR is disabled, traffic is delivered again after 86 ms. As rerouting without FRR requires controller interaction, the measured restoration time depends on controller load, network size, and communication delay. In this experiment, there is only a single flow affected by the faiure, the overall network is small despite the additional network, and the controller is directly connected to the Tofino. Therefore, the experimental result for the restoration time is likely lower than restoration times in production networks.

If FRR is enabled, traffic is delivered after a small restoration time of 0.6 ms. Here, the switchover from primary egress port to backup egress port at the Tofino is independent of controller load, network size, and communication delay as FRR is a switch-local mechanism. Thus, restoration times can be greatly reduced by FRR on P4-capable hardware. Moreover, the mechanism is general enough to support all RoLPS protection variants by appropriate configuration through the controller.

C. Loop Detection

We experimentally evaluate the capability of ALD to detect and stop loops. We present the modified testbed, explain two different experiments and the studied metric, and finally we discuss measurement results.

1) Testbed: Figure 11 shows the testbed. The Tofino is

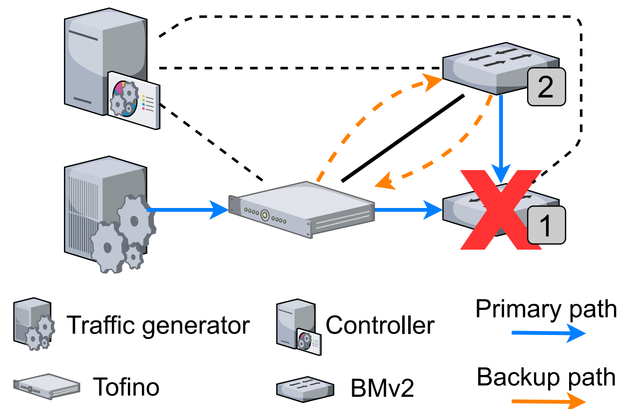


Figure 11: Testbed for evaluation of ALD.

now conntected to two BMv2s (BMv2-1, BMv2-2) which are also connected with each other. The controller configures the Tofino and all BMv2s with available LP-LFAs upon startup. In the experiments, the traffic generator sends a packet towards BMv2-1. The Tofino has BMv2-2 as an LFA when BMv2-1 is not reachable. Likewise, BMv2-2 has the Tofino as an LFA when BMv2-1 is not reachable. If BMv2-1 fails, traffic destined to that node loops between the Tofino and BMv2-2. However, the TTL in the IP header is set to 64 when sent by the traffic generator and decremented whenever forwarded by a node. The packet is dropped when its TTL reached 0.

2) Experiments and Metric: We perform two experiments with ALD disabled and ALD enabled on the switches. We track packet arrivals at BMv2-2 using tcpdump. Thereby we can observe how often a looping packet is received.

3) Results: Figure 12 illustrates a log of packet arrivals at BMv2-2, starting with time 0 at first packet arrival. Without

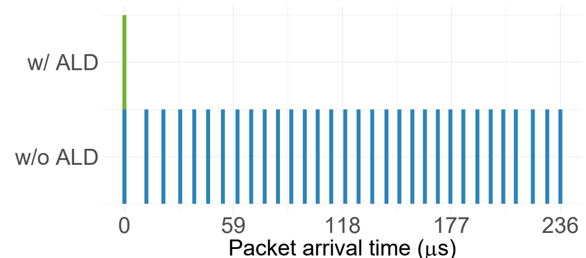


Figure 12: Packet arrivals at BMv2-2 without and with ALD.

ALD, BMv2-2 receives the packet 32 times. Thus, the packet looped between the Tofino and BMv2-2 until it was dropped due to TTL=0. With ALD, BMv2-2 receives the packet only

once. It then redirects the packet to the Tofino which then drops the packet at the attempt to reroute the packet for the third time. Therefore, BMv2-2 receives the packet only once.

VIII. Conclusion

In this paper we presented robust LFA protection for software-defined networks (RoLPS). It leverages loop-free alternates (LFAs) and remote LFAs (rLFAs) known from IP networks to forward traffic over alternative next-hops if primary next-hops are not reachable. However, this alone cannot protect all destinations against failures and may cause forwarding loops under challenging conditions. Therefore, we proposed explicit LFAs (eLFAs) using explicit rerouting tunnels to cover all destinations. eLFAs are conceptually similar to topology-independent LFAs (TI-LFAs) but do require only a single additional header segment for protection while protection with typical TI-LFAs may require a clearly larger header stack. Furthermore we describe advanced loop detection (ALD) to stop forwarding loops. These mechanisms are simple and do not require controller interaction. We suggested various protection variants that utilize (e/r)LFAs with different protection quality and complexity.

We evaluated RoLPS through simulations based on 208 representative topologies. The results revealed that existing (r)LFAs cannot provide all destinations and lead to substantial forwarding loops in case of node failures. More elaborate RoLPS variants with eLFAs and ALD, e.g., ALD-NP-eLFA, protect all traffic against all single link or node failures in networks with both unit and non-unit link costs. Furthermore, they protect most destinations against multiple failures ($> 90\%$) and prevent forwarding loops. A drawback of eLFAs is that they required additional forwarding entries. However, our evaluation showed that RoLPS protection variants require only very few eLFAs, in particular compared to other FRR mechanisms such as MPLS facility backup, MRTs, MRCs, IDAGs, or not-via addresses. Thus, the full protection coverage against single link or node failures together with the need for only a few additional forwarding entries make RoLPS attractive for software-defined networks. In addition, RoLPS protection variants extends lengths of backup paths compared to those of shortest path recomputation, but there is no visible difference to backup path lengths with MPLS facility backup.

We implemented a P4-based prototype that features RoLPS-based protection variants. The source code is publicly available. A measurement study showed that the prototype achieves a throughput of 100 Gb/s, restores connectivity in less than 1 ms including failure detection, and reliably detects and stops forwarding loops.

Acknowledgment

We acknowledge the support from BelWü for borrowed high-performance hardware for the measurement-based evaluations. Likewise, we appreciate the work of Irene Müller-Benz for the development of an early prototype of RoLPS.

Acronyms and Glossary

FRR	fast reroute
PLR	point of local repair
LFA	loop-free alternate [20]
rLFA	remote LFA [23], [24]
eLFA	explicit LFA [2]
TI-LFA	topology-independent LFA [3]
MPLS	multiprotocol label switching [8]
MRT	maximally redundant tree [15]
IDAG	independent directed acyclic graph [18]
MRC	multiple routing configuration [14]
SLF	single link failure
SNF	single node failure
DLF	double link failure
LP	link protecting
NP	node protecting
ALD	advanced loop detection
RoLPS	robust LFA protection for SDN

Table 3: Acronyms.

Point of local repair (PLR)	A node that cannot forward a packet to the default next-hop because of a failure. It executes precomputed backup actions to locally reroute packets around the failure.
Loop-free alternate (LFA)	Alternative next-hop that successfully forwards failure-affected traffic towards the destination. Simple LFAs cannot protect all destinations.
rLFA	Remote nodes in the network that successfully forward traffic towards the destination. PLRs reach rLFAs through shortest path tunnels. rLFAs protect more destinations than LFAs. However, they cannot protect all destinations against SLF in non-unit link cost networks or SNF in general.
eLFA	Similar to rLFAs. However, PLRs reach eLFAs through explicit tunnels implemented by additional forwarding entries. eLFAs protect against all SLF and SNF independent of link costs. Multipoint-to-point tunnels reduce the number of additional forwarding entries.
Link protecting (LP)	A link protecting (e/r)LFA avoids the link between PLR and next-hop. They may cause rerouting loops for SNF.
Node protecting (NP)	A node protecting (e/r)LFA avoids the next-hop. There are significantly less NP-(e/r)LFAs than LP-(e/r)LFAs. NP implies LP, i.e., it is the stronger property.
Loop detection (LD) [1]	A mechanism to detect and stop rerouting loops caused by LFAs. May erroneously drop packets.
LD-LFA [1]	LD-LFA preferably uses NP-LFAs for protection. Only when no NP-LFA is available, LP-LFAs are used to increase the number of protected destinations. In addition, LD-LFA leverages loop detection to prevent loops.
Advanced loop detection (ALD)	A mechanism to detect and stop loops caused by LFAs. Allows to reroute a packet two times to cope with double failures.
Robust LFA protection for SDN (RoLPS)	Protection concept presented in this paper. It defines eLFAs and ALD. RoLPS ranks (e/r)LFAs and selects the best one. Uses ALD to detect and stop loops.

Table 4: Glossary.

References

[1] W. Braun and M. Menth, “Loop-Free Alternates with Loop Detection for Fast Reroute in Software-Defined Carrier and Data Center Networks,” *Journal of Network and Systems Management*, vol. 24, 2016.

[2] D. Merling, W. Braun, and M. Menth, “Efficient Data Plane Protection for SDN,” in *IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2018.

[3] P. Francois, C. Filsfils, A. Bashandy, B. Decraene, and S. Litkowski, *Topology Independent Fast Reroute using Segment Routing*, https://tools.ietf.org/html/draft-ietf-rtgwg-segment-routing-ti-lfa-06, Feb. 2021.

[4] S. Rai, B. Mukherjee, and O. Deshpande, “IP Resilience within an Autonomous System: Current Approaches, Challenges, and Future Directions,” *IEEE Communications Magazine*, vol. 43, 2005.

[5] A. Raj and O. Ibe, “A Survey of IP and Multiprotocol Label Switching Fast Reroute Schemes,” *Computer Networks*, vol. 51, no. 8, 2007.

[6] J. Papan, P. Segeč, P. Palúch, and L. Mikus, “The Survey of Current IPFRR Mechanisms,” in *Federated Conference on Software Development and Object Technologies*, Dec. 2017.

[7] D. Hutchison and J. P. Sterbenz, “Architecture and design for resilient networked systems,” *Computer Communications*, vol. 131, 2018.

[8] E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol Label Switching Architecture*, https://tools.ietf.org/html/rfc3031, Jan. 2001.

[9] Ping Pan and George Swallow and Alia Atlas, *RFC4090: Fast Reroute Extensions to RSVP-TE for LSP Tunnels*, https://tools.ietf.org/html/rfc4090, May 2005.

[10] K. Kompella and W. Lin, *No Further Fast Reroute*, https://tools.ietf.org/html/draft-kompella-mpls-nffrr-00, Mar. 2020.

[11] S. Bryant, S. Previdi, and M. Shand, *RFC6981: A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses*, http://www.rfc-editor.org/rfc/rfc6981.txt, Jul. 2013.

[12] R. Martin, M. Menth, M. Hartmann, T. Cicic, and A. Kvalbein, “Loop-Free Alternates and Not-Via Addresses: A Proper Combination for IP Fast Reroute?” *Computer Networks*, vol. 54, 2010.

[13] S. Nelakuditi *et al.*, “Fast Local Rerouting for Handling Transient Link Failures,” *IEEE/ACM Trans. on Networking*, Apr. 2007.

[14] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, “Fast IP Network Recovery Using Multiple Routing Configurations,” in *IEEE Infocom*, Apr. 2006.

[15] A. Atlas, C. Bowers, and G. Enyedi, *RFC7812: An Architecture for IP/LDP Fast Reroute Using Maximally Redundant Trees (MRT-FRR)*, http://www.rfc-editor.org/rfc/rfc7812.txt, Jun. 2016.

[16] M. Menth and W. Braun, “Performance Comparison of Not-Via Addresses and Maximally Redundant Trees (MRTs),” in *IEEE/IFIP IM*, Apr. 2013.

[17] K. Kuang, S. Wang, and X. Wang, “Discussion on the Combination of Loop-Free Alternates and Maximally Redundant Trees for IP Networks Fast Reroute,” in *IEEE International Conference on Communications*, Jun. 2014.

[18] S. Cho, T. Elhourani, and S. Ramasubramanian, “Independent Directed Acyclic Graphs for Resilient Multipath Routing,” *IEEE/ACM Transactions on Networking*, vol. 20, Feb. 2012.

[19] S. S. Lor, R. Landa, and M. Rio, “Packet re-cycling: Eliminating packet losses due to network failures,” in *ACM Workshop on Hot Topics in Networks*, 2010.

[20] A. Atlas and A. Zinin, *RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates*, http://www.rfc-editor.org/rfc/rfc5286.txt, 2008.

[21] L. Csikor, M. Nagy, and G. Rétvári, “Network Optimization Techniques for Improving Fast IP-level Resilience with Loop-Free Alternates,” *Infocommunications Journal*, vol. 3, 2011.

[22] L. Csikor, J. Tapolcai, and G. Retvari, “Optimizing IGP link costs for improving IP-level resilience with Loop-Free Alternates,” *Computer Communications*, vol. 36, 2013.

[23] S. Bryant, C. Filsfils, S. Previdi, M. Shand, and N. So, *RFC7490:Remote Loop-Free Alternate (LFA) Fast Reroute (FRR)*, https://tools.ietf.org/html/rfc7490, 2015.

[24] P. Sarkar, S. Hegde, C. Bowers, H. Gredler, and S. Litkowski, *Remote-LFA Node Protection and Manageability*, https://tools.ietf.org/html/rfc8102, 2017.

[25] L. Csikor and G. Retvari, “On Providing Fast Protection with Remote Loop-Free Alternates: Analyzing and Optimizing Unit Cost Networks,” in *Telecommunication Systems*, 2015.

[26] G. Retvari, J. Tapolcai, G. Enyedi, and A. Csaszar, “IP Fast ReRoute: Loop Free Alternates Revisited,” in *IEEE Infocom*, Apr. 2011.

[27] W. Tavernier, D. Papadimitriou, D. Colle, M. Pickavet, and P. Demeester, “Self-configuring Loop-free Alternates with High Link Failure Coverage,” *Telecommunication Systems*, vol. 56, 2014.

[28] A. Farrel and R. Bonica, “Segment Routing: Cutting Through the Hype and Finding the IETF’s Innovative Nugget of Gold,” *IETF Journal*, vol. 13, 2017.

[29] Y. E. Oktian *et al.*, “Distributed SDN Controller System: A Survey on Design Choice,” *Computer Networks*, vol. 121, 2017.

[30] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “OpenFlow: Meeting Carrier-Grade Recovery Requirements,” *Computer Communications*, vol. 36, 2013.

[31] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, “Resilience support in software-defined networking: A survey,” *Computer Networks*, vol. 92, 2015.

[32] M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, *A Survey of Fast Recovery Mechanisms in the Data Plane*, https://www.techrxiv.org/articles/preprint/Fast_Recovery_Mechanisms_in_the_Data_Plane/12367508/2, May 2020.

[33] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takàcs, and P. Sköldström, "Scalable Fault Management for OpenFlow," in *IEEE International Conference on Communications*, 2012.

[34] N. L. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast Recovery in Software-Defined Networks," in *European Workshop on Software Defined Networks*, Sep. 2014.

[35] R. M. Ramos *et al.*, "SlickFlow: Resilient Source Routing in Data Center Networks Unlocked by OpenFlow," in *IEEE Conference on Local Computer Networks*, Oct. 2013.

[36] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sansó, "SPIDER: Fault Resilient SDN Pipeline with Recovery Delay Guarantees," in *IEEE Conference on Network Softwarization*, Jun. 2016.

[37] N. L. M. van Adrichem, F. Iqbal, and F. A. Kuipers, "Backup Rules in Software-Defined Networks," in *IEEE Conference on Network Function Virtualization and Software-Defined Networking*, Nov. 2016.

[38] S. Cevher, M. Ulutas, S. Altun, and I. Hokelek, "Multi Topology Routing Based IP Fast Re-Route for Software Defined Networks," in *IEEE Symposium on Computers and Communications*, Jun. 2016.

[39] S. Cevher, "Multi Topology Routing Based Failure Protection for Software Defined Networks," in *IEEE International Black Sea Conference on Communications and Networking*, Jun. 2018.

[40] Q. Li, Y. Liu, Z. Zhu, H. Li, and Y. Jiang, "BOND: Flexible failure recovery in software defined networks," *Computer Networks*, vol. 149, 2019.

[41] R. Sedar, M. Borokhovich, M. Chiesa, G. Antichi, and S. Schmid, "Supporting Emerging Applications With Low-Latency Failover in P4," in *Workshop on Networking for Emerging Applications and Technologies*, 2018.

[42] H. Giesen, L. Shi, J. Sonchack, A. Chelluri, N. Prabhu, N. Sultana, L. Kant, A. J. McAuley, A. Poylisher, A. DeHon, and B. T. Loo, "In-Network Computing to the Rescue of Faulty Links," in *Morning Workshop on In-Network Computing*, 2018.

[43] S. Lindner, D. Merling, M. Häberle, and M. Menth, "P4-Protect: 1+1 Path Protection for P4," *P4 Workshop in Europe (EuroP4)*, Dec. 2020.

[44] K. Hirata and T. Tachibana, "Implementation of Multiple Routing Configurations on Software-Defined Networks with P4," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2019.

[45] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, *D2R: Dataplane-Only Policy-Compliant Routing Under Failures*, 2019.

[46] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, and S. Schmid, "PURR: A Primitive for Reconfigurable Fast Reroute," in *ACM Conference on emerging Networking EXperiments and Technologies*, 2019.

[47] C. Filsfils *et al.*, *RFC8986: Segment Routing over IPv6 (SRv6) Network Programming*, https://www.rfc-editor.org/rfc/rfc8986.txt, 2021.

[48] A. Bashandy *et al.*, *RFC8660: Segment Routing with the MPLS Data Plane*, https://www.rfc-editor.org/rfc/rfc8660.txt, 2019.

[49] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, 2011.

[50] S. Halabi, *OSPF DESIGN GUIDE*, http://rtfm.vtt.net/spf1euro.pdf, 1996.

[51] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM CCR*, vol. 44, 2014.

[52] p4lang, *Behavioral-model*, https://github.com/p4lang/behavioral-model, 2019.

[53] Edge-Core Networks, *The World's Fastest & Most Programmable Networks*, https://barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/, 2017.

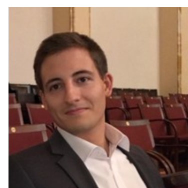
[54] ——, *Wedge100BF-32X/65X Switch*, https://www.edge-core.com/_upload/images/Wedge100BF-32X_65X_DS_R05_20191210.pdf, 2019.

[55] D. Merling, S. Lindner, and M. Menth, "Hardware-Based Evaluation of Scalable andResilient Multicast with BIER in P4," *IEEE Transactions on Network and Service Management*, In Revision for TNSM special issue: Advanced Management of Softwarized Networks.

[56] EXFO, *FTB-1v2/FTB-1 Pro Platform*, https://www.exfo.com/umbraco/surface/file/download/?ni=10900&cn=en-US&pi=5404, 2019.

[57] A. Bas, *BMv2 Throughput*, https://github.com/p4lang/behavioral-model/issues/537#issuecomment-360537441, Jan. 2018.

Daniel Merling is a Ph. D. student at the chair of communication networks of Prof. Dr. habil. Michael Menth at the Eberhard Karls University Tuebingen, Germany. There he obtained his master's degree in 2017 and afterwards, became part of the communication networks research group. His area of expertise include software-defined networking, scalability, P4, routing and resilience issues, multicast and congestion management.



Steffen Lindner is a Ph.D. student at the Eberhard Karls University Tübingen, Germany. He wrote his bachelor and master thesis at the chair of communication networks of Prof. Dr. habil. Michael Menth. He started his Ph.D. in September 2019 at the communication networks research group. His research interests include software-defined networking, P4 and congestion management.

Michael Menth (Senior Member, IEEE) is professor at the Department of Computer Science at the University of Tuebingen/Germany since 2010 and chairholder of Communication Networks. He studied, worked, and obtained diploma (1998), PhD (2004), and habilitation (2010) degrees at the universities of Austin/Texas, Ulm/Germany, and Wuerzburg/Germany. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, industrial networking and Internet of Things, software-defined networking and Internet protocols.

1.2 P4-Protect: 1+1 Path Protection for P4

P4-Protect: 1+1 Path Protection for P4

Steffen Lindner
University of Tübingen
steffen.lindner@uni-tuebingen.de

Daniel Merling
University of Tübingen
daniel.merling@uni-tuebingen.de

Marco Häberle
University of Tübingen
marco.haeberle@uni-tuebingen.de

Michael Menth
University of Tübingen
menth@uni-tuebingen.de

ABSTRACT

1+1 protection is a method to secure traffic between two nodes against failures in between. The sending node duplicates the traffic and forwards it over two disjoint paths. The receiving node assures that only a single copy of the traffic is further forwarded to its destination. In contrast to other protection schemes, this method prevents almost any packet loss in case of failures. 1+1 protection is usually applied on the optical layer, on Ethernet, or on MPLS.

In this work we propose the application of 1+1 for P4-based IP networks. We define an 1+1 protection header for that purpose. We describe the behavior of sending and receiving nodes and provide a P4-based implementation for the Behavioral Model version 2 (bmv2) software switch and the hardware switch Tofino Edgecore Wedge 100BF-32X. We illustrate how to secure traffic, e.g. individual TCP flows, on the Internet with this approach. Finally, we present performance results showing that the P4-based implementation efficiently works on the Tofino Edgecore Wedge 100BF-32X.

KEYWORDS

p4, software defined networking, 1+1 protection

ACM Reference Format:
Steffen Lindner, Daniel Merling, Marco Häberle, and Michael Menth. 2020. P4-Protect: 1+1 Path Protection for P4. In *3rd P4 Workshop in Europe (EuroP4'20), December 1, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3426744.3431327

1 INTRODUCTION

There are various concepts to secure traffic transmission against failure of path components such as links or nodes. The fastest is 1+1 protection. A sender duplicates traffic and forwards it over disjoint paths while the receiver forwards only the first copy received for every packet. In case of a failure, any packet loss can be avoided, which makes 1+1 protection attractive for highly reliable applications. 1+1 protection is implemented in optical networks to protect an entire trunk. It is also available for MPLS [10] and Ethernet [9], which are carrier technologies for IP and introduce signaling complexity. In this paper, we leverage the P4 programming language [3] to provide 1+1 protection for IP networks. We program P4 switches such that they feature IP forwarding, the sending and receiving node behaviour of 1+1 protection which includes IP encapsulation and decapsulation. We call this approach P4-Protect. Targets of our implementation are the software switch BMv2 and the hardware switch Tofino Edgecore Wedge 100BF-32X. A particular challenge is the selection of the fist copy of every duplicated packet at the receiver. We provide a controller that allows to set up 1+1 protection between P4 nodes implementing P4-Protect. Furthermore, protected flows can be added using a fine-granular description based on various header fields. We evaluate the performance of P4-Protect on the hardware switch. We show that P4-Protect can be used with only marginal throughput degradation and we illustrate that P4-Protect can significantly reduce jitter when both paths have similar delays.

The paper is structured as follows. Section 2 gives an overview of related work. Section 3 describes the 1+1 protection mechanism used for our implementation and extensions for its use on the general Internet. Section 5 presents a P4-based implementation including specifics for the Tofino Edgecore Wedge 100BF-32X. We evaluate the performance of P4-Protect on the hardware switch in Section 6 and conclude the paper in Section 7.

2 RELATED WORK

We review various resilience concepts for communication networks. Afterwards, we give examples for 1+1 protection.

2.1 Overview

Rerouting reorganizes the traffic forwarding to avoid failed components. This happens on a time scale of a second. Fast reroute (FRR) locally detects that a next hop is unreachable and deviates traffic to an alternative next hop [1]. The detection may take a few 10s of milliseconds so that traffic loss cannot be avoided. Both rerouting and FRR do not utilize backup resources under failure-free conditions, but their reaction time suffers from failure detection delay. 1:1 protection leverages a primary/backup path concept. To switch over, the head-end node of the paths needs to be informed about a failure, which imposes additional delay. With restoration, recovery paths may be dynamically allocated so that even more time is needed to establish the restoration paths [19, p. 31]. 1+1 protection duplicates traffic and sends it over two disjoint paths whereby the receiving node needs to eliminate duplicates. That method is fastest, but it requires extra capacities also under failure-free conditions. Some services can afford short network downtimes, other services greatly benefit from 1+1 protection's high reliability.

The surveys [15], [20], and [7] provide an overview of various protection and restoration schemes. The authors of [7] discuss survivability techniques for non-WDM networks like automatic protection switching (APS) and self healing rings (SHR) as well

EuroP4'20, December 1, 2020, Barcelona, Spain
© 2020 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *3rd P4 Workshop in Europe (EuroP4'20), December 1, 2020, Barcelona, Spain*, https://doi.org/10.1145/3426744.3431327.

as dynamic restoration schemes in SONET. They further describe protection methods for optical WDM networks. A comprehensive overview of protection and restoration mechanisms for optical, SONET/SDH, IP, and MPLS networks can be found in [19].

SDN with inband signalling increases the need for fast and local protection against failures because the controller may no longer be reachable in case of a failure or highly loaded. In addition, with SDN new protection mechanisms can be implemented, e.g., to reduce state in the network. Examples are given in [14].

2.2 1+1 Protection

At first we will look at standards with respect to 1+1 protection, followed by other work related to 1+1 protection.

2.2.1 Standards. The ITU-T specification Y.1703 [10] defines a 1+1 path protection scheme for MPLS. It adds sequence numbers to packets and replicates them on disjoint paths. At the end of the paths, duplicate packets are identified by the sequence number and eliminated. P4-Protect works similarly. However, it does not require MPLS. It is compatible with IP and works over the Internet.

802.1CB [8] defines a redundant transmission mode for Time-Sensitive Networking (TSN), called *Frame Replication and Elimination for Reliability* (FRER). Each packet of a stream is equipped with a sequence number, replicated, and then sent through two disjoint paths to a destination. Both destination and/or traversing nodes eliminate duplicate packets. FRER supports two algorithms: *VectorRecoveryAlgorithm* and *MatchRecoveryAlgorithm*. With the *VectorRecoveryAlgorithm*, an acceptance window is used to accept packets with higher sequence numbers than expected. With the *MatchRecoveryAlgorithm* all sequence numbers except the last seen are accepted, which is used to prevent misbehaviour. In-order delivery is currently out of scope in 802.1CB.

DetNet [5] provides capabilities to carry data flows for real-time applications with extremely low data loss rates and bounded latency within a network. *Packet Replication and Elimination* (PRE) is a service protection method for DetNet, which leverages the 1+1 protection concept. PRE adds sequence numbers or time stamps to packets in order to identify duplicates. Packets are replicated and sent along multiple different paths, e.g., over explicit routes. Duplicates are eliminated, mostly at the edge of the DetNet domain. The *Packet Ordering Function* can be used at the elimination point to provide in-order delivery. However, this requires extra buffering.

2.2.2 Other Work on 1+1 Protection. The authors of [21] compare several implementation strategies of 1+1 protection, i.e, traditional 1+1 path protection, network redundancy 1+1 path protection (diversity coding) [2], and network-coded 1+1 path protection. Their analytical results show that diversity coding and network coding can be more cost-efficient, i.e., they require about 5-20% less reserved bandwidth. The delay impact of 1+1 path protection in MPLS networks has been investigated in [17]. McGettrick et. al [13] consider 10 Gb/s symmetric LR-PON. They reveal switch-over times to a backup OLT of less than 4 ms. Multicast traffic has often real-time requirements. Mohandespour et. al extend the idea of unicast 1+1 protection to protect multicast connections [16]. They formulate the problem of minimum cost multicast 1+1 protection as a 2-connectivity problem and propose heuristics. Braun et. al [4] propose maximally redundant trees for 1+1 protection in BIER, a stateless multicast transport mechanism. It leverages the concept of multicast-only FRR [11].

3 P4-PROTECT: CONCEPT

We first give an overview of P4-Protect. We present its protection header, the protection connection context, and the operation of the Protection Tunnel Ingress (PTI) and Protection Tunnel Egress (PTE).

3.1 Overview

With P4-Protect, a protection connection is established between two P4 switches. Protected traffic is duplicated by a PTI node and simultaneously carried through two protection tunnels to a PTE node. The PTE receives the duplicated traffic and forwards the first copy received for every packet.

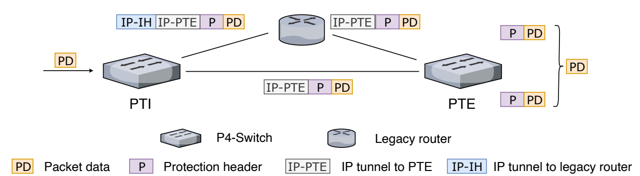


Figure 1: With P4-Protect, a PTI encapsulates and duplicates packets, and sends them over disjoint paths; the PTE decapsulates the packets and forwards only the first packet copy.

Figure 1 illustrates the protocol stack used with P4-Protect. The PTI adds to each packet received for a protected flow a protection header (P) that contains a sequence number which is incremented for each protected packet. The packet is equipped with an additional IP header (IP-PTE) with the PTE's IP address as destination. The PTI duplicates that packet and forwards the two copies over different paths. The paths may be different due to traffic engineering (TE) capabilities of the network or path diversity may be achieved through an additional intermediate hop. When the PTE receives a packet, it removes its outer IP header (IP-PTE). If the sequence number in the protection header is larger than the last sequence number received for this connection, it removes the protection header and forwards the packet; otherwise, the packet is dropped. The latter is needed as duplicate packets are also considered harmful.

3.2 Protection Header

The protection header contains a 24 bit Connection Identifier (CID), a 32 bit Sequence Number (SN) field, and an 8 bit *next protocol* field. The CID is used to uniquely identify a protection connection at the PTE. The sequence number is used at the PTE to identify duplicates. The *next protocol* field facilitates the parsing of the next header. We reuse the IP protocol numbers for this purpose.

3.3 Protection Connection Context

A protection connection is set up between a PTI and PTE. Their IP addresses are associated with this connection, including two interfaces over which duplicate packets are forwarded. For each connection, the PTI has a sequence number counter SN_{last}^{PTI} which

is incremented for each packet forwarded over the respective protection connection. Likewise, the PTE has a variable SN_{last}^{PTE} which records the highest sequence number received for the respective protection connection. A CID is used to identify a connection at the PTE. A PTI may have several protection connections with the same CID but different PTEs (see Section 5.5.1).

3.4 PTI Operation

The PTI has a set of flow descriptors that are mapped to protection connections. If the PTI receives a packet which is matched by a specific flow descriptor, the PTI processes the packet using the corresponding protection connection. That is, it increments the SN_{last}^{PTI}, adds a protection header with CID, *next protocol* set to `IPv4`, and the SN set to SN_{last}^{PTI}. Then, an IP header is added using the PTI's IP address as source and the IP address of the PTE associated with the protection connection as destination. The packet is duplicated and forwarded over the two paths associated with the protection connection.

3.5 PTE Operation

During failure-free operation, the PTE receives duplicate packets via two protection tunnels. When the PTE receives a packet, it decapsulates the outer IP header. It uses the CID in the protection header to identify the protection connection and the corresponding SN_{last}^{PTE}. If the SN in the protection header is larger than SN_{last}^{PTE}, SN_{last}^{PTE} is updated by SN, the protection header is decapsulated, and the original packet is forwarded; otherwise, the packet is dropped.

The presented behavior works for unlimited sequence numbers. The limited size of the sequence number space makes the acceptance decision for a packet more complex. Then, a SN larger than SN_{last}^{PTE} may indicate a copy of a new packet, but it may also result from a very old packet. To solve this problem, we adopt the use of an acceptance window as proposed in [10]. The window is W sequence numbers large. Let SN_{max} be the maximum sequence number. If $SN_{last}^{PTE}+W < SN_{max}$ holds, a new sequence number SN is accepted if the following inequality holds:

$$SN_{last}^{PTE} < SN \leq SN_{last}^{PTE} + W \quad (1)$$

If $SN_{last}^{PTE} + W \geq SN_{max}$ holds, a new sequence number SN is accepted if one of the two following inequalities holds:

$$SN_{last}^{PTE} < SN \quad (2)$$

$$SN < SN_{last}^{PTE} + W - SN_{max} \quad (3)$$

This allows a packet copy to arrive $SN_{max} - W$ sequence numbers later than the corresponding first packet copy without being recognized as new packets.

AXE [12] tries to solve a similar problem, namely the de-duplication of packets. The hash of incoming packets is used to access special registers and associated header fields are stored. When another packet with the same hash arrives and the stored header fields match the incoming packet, the packet is a duplicate. No hash collision is considered. This technique detects duplicates quite reliably. However, AXE considers L2 flooding for learning bridges and therefore operates on relatively low bandwidths. P4-Protect must be able to de-duplicate several 100G connections, for the Tofino Edgecore Wedge 100BF-32X 3.2 Tb/s. Hence the AXE approach is not feasible due to the required register memory space and, depending on the hash algorithm, the high probability for hash collisions.

4 DISCUSSION

In this section the protection properties of P4-Protect are examined in more detail. Both, advantages and limitations of P4-Protect are discussed. The impact on jitter, packet loss and packet reordering are considered. To that end, we provide examples of traffic streams received by the PTE and their results after duplicate elimination.

4.1 Impact on Jitter

P4-Protect replicates packets to two preferable disjoint paths. If both paths suffer from jitter, P4-Protect can compensate the overall end-to-end jitter. Figure 2 illustrates the impact of P4-Protect on the overall end-to-end jitter. Packet 3 on path 1 has a very high delay due to jitter. As P4-Protect always forwards the first version of in-order packets, packet 3 is forwarded from the second path and thereby compensates the jitter delay.

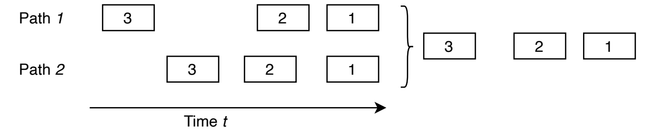


Figure 2: P4-Protect can reduce jitter.

4.2 Impact of Packet Loss

P4-Protect forwards the first version of a packet. If a path fails, all packet replicas of the other path are forwarded correctly. If individual packets are lost on one path, their replicas from the other path are not necessarily forwarded. This phenomenon ist illustrated in Figure 3. Four packets are replicated by the PTI and sent over two disjoint paths. The second path has a higher latency. As a result, packet 4 of the first path arrives before packet 3 on the second path. Now, SNE_{last}^{PTE} is set to 4, and as a consequence, packet 3 on the second path is discarded.

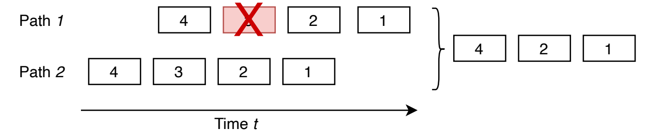


Figure 3: Packet loss may not be compensated by P4-Protect.

This behavior is due to the scalable design of P4-Protect. Only the last accepted sequence number is stored and checked at a new packet arrival. Missing packets are not memorized nor are packets buffered. This example clarifies that the objective of P4-Protect is to protect quickly against path failures, it is not to compensate for individual packet losses.

4.3 Packet Reordering

Packet reordering on a path has different sources, e.g., parallelism in network devices, link bundling, and special QoS configurations [18]. In case of packet reordering, P4-Protect may cause packet loss.

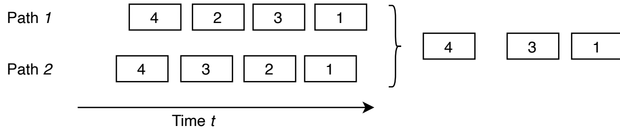


Figure 4: As it is not possible to check for lost packets, reordering leads to packet loss.

Figure 4 illustrates the impact of packet reordering. Path 1 has a slightly lower end-to-end latency than path 2. Due to packet reordering, the PTE receives the packets of path 1 in the order 1 3 2 4 instead of 1 2 3 4 . Moreover, packet 3 of the first path arrives slightly before packet 2 of the second path. As a result, the PTE accepts packets 1, 3 and 4 from path 1 and discards packet 2 from path 2. The main reason for this behavior is that P4-Protect does not memorize missing packets. Therefore, they cannot be accepted if they arrive in the wrong order. Limited arithmetic operations and storage access on our specific hardware target inhibit more sophisticated checks.

5 IMPLEMENTATION

In this section we present the implementation of P4-Protect. We describe the supported header stacks, explain the control blocks, their organization in ingress and egress control flow, and we reveal implementation details about some control blocks. Finally, we sketch most relevant aspects of the P4-Protect controller.

5.1 Supported Header Stacks

Incoming packets are parsed so that their header values can be accessed within the P4 pipeline. To that end, we define the following supported header stacks. Unprotected IP traffic has the structure IP/TP, i.e., IP header and some transport header (TCP/UDP), and protected IP traffic has the structure IP/P/IP/TP, i.e., the IP header with the PTE's address, the protection header, the original IP header, and a transport header. IP traffic without transport header is parsed only up to the IP header.

5.2 Control Blocks

We present three control blocks of our implementation of P4-Protect. They consider the packet processing by PTI and PTE.

5.2.1 Control Block: Protect&Forward. When the PTI receives an IP packet, it is parsed and matched against the Match+action (MAT) table ProtectedFlows. In case of a match, the packet is equipped with an appropriate header stack, duplicated, and sent to appropriate egress ports. In case of a miss, the packet is processed by a standard IPv4 forwarding procedure.

5.2.2 Control Block: Decaps-IP. When the PTE receives an IP packet with the PTE's own IP address, the IP header is decapsulated. If the next protocol indicates a protection header, the packet is handed over to the Decaps-P control block; otherwise, the packet is processed by the Protect&Forward control block since the resulting packet may need to be protected and forwarded.

5.2.3 Control Block: Decaps-P. In the Decaps-P control block, the PTE examines the protection header and decides whether to keep or drop the packet as it is a copy of an earlier received packet. To keep the packet, the protection header is decapsulated.

5.3 Ingress and Egress Control Flow

The inter-dependencies between the control blocks suggest the following ingress control flow: Decaps-IP, Decaps-P, Protect&Forward. At a mere PTI, no action is performed by the Decaps-IP and Decap-P control block. The Protect&Forward takes care that protected traffic is duplicated and sent over two different paths and that unprotected traffic is forwarded by normal IPv4 operation. At a mere PTE, protected traffic is decapsulated and selected before being forwarded by normal IPv4 operation. Unprotected traffic is just forwarded by normal IPv4 operation.

5.4 Control Block Implementations

In the following, we explain implementation details of the Protect&Forward control block and the Decaps-P control block. We omit the Decaps-IP control block as it is rather simple.

5.4.1 Protect&Forward Control Block. The operation of the Protect&Forward control block is illustrated in Figure 5. It utilizes the MAT ProtectedFlows to process all packets. It effects that protected traffic is encapsulated at the PTI with a protection header and an IP header for tunneling.

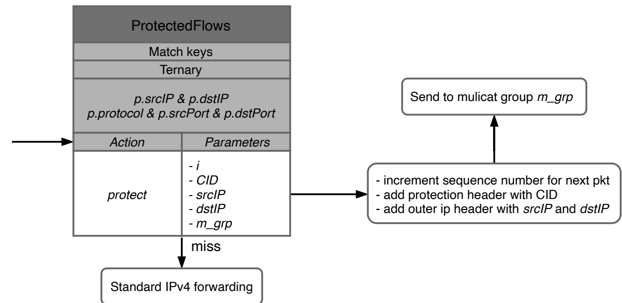


Figure 5: The MAT ProtectedFows inside the Protect&Forward control block is applied to IPv4 traffic.

The MAT ProtectedFlows uses a ternary match on the classic 5-tuple description of a flow: the source and destination IP address and port as well as the protocol field. In case of a match, the MAT maps a packet to a specifc protection connection and calls the protect action with the connection-specific parameters i, CID, $srcIP$, $dstIP$, and m_grp. The protect action increments the register $SN^{PTI}_{last}[i]$ where i is a connection-specific index to access a register containing the last sequence number. On the Tofino target, this is performed by a separate register action. The protect action further pushes a protection header on the packet including CID, i.e., the

CID, $SN_{last}^{PTI}[i]$, and the next protocol set to `IPv4`. Then, it pushes an IPv4 header with the IP address *srcIP* of the PTI as source IP and the IP address *dstIP* of the PTE as destination IP. The protocol field of this outer IP header is set to `P4-Protect`. Finally, the multicast group of the packet is set to *m_grp*. It is a connection-specific multicast group. It effects that the packet is duplicated and sent to two egress ports in order to deliver it via two protection tunnels to the PTE. In case of a miss, the packet is unprotected and handled by a standard IPv4 forwarding procedure, which is not further explained in this paper.

5.4.2 Decaps-P Control Block. The Decaps-P control block decides whether a packet is new and should be forwarded or dropped. It compares the sequence number SN of the packet's protection header with the last sequence number of the corresponding protection connection. The latter can be accessed by the register $SN_{last}^{PTE}[CID]$ where CID is given in the protection header. The acceptance is decided based on Equation (1) or Equation (3) depending on the value of SN and W where W is given as a constant.

As the check is rather complex, it requires careful implementation for the Tofino target [1]. It leverages the fact that we set $W = \frac{SN_{max}}{2}$. Furthermore, it requires a reformulation of Equation (1) and Equation (3).

If $W \leq SN$ holds, the following two inequalities must be met:

$$SN_{last}^{PTE} < SN \quad (4)$$

$$SN - SN_{last}^{PTE} \leq W \quad (5)$$

Otherwise, if $SN < W$, it is sufficient that only one of the following two inequalities holds:

$$SN_{last}^{PTE} < SN \quad (6)$$

$$W \leq SN_{last}^{PTE} - SN \quad (7)$$

Both cases are implemented as separate register actions on the Tofino target. With 32 bit sequence numbers, a minimum packet size of 40 bytes and a transmission speed of $C = 1$ Tb/s, a delay difference up to 1.6s can be compensated.

The bmv2 version of the implementation can be found at Github[2]. The Tofino version of the implementation can be found at Github[3] as well.

5.5 Controller for P4-Protect

P4-Protect's controller offers an interface for the management of protection connections and protected flows. It configures in particular the MAT ProtectedFlows but also other MATs needed for standard IPv4 forwarding or IP decapsulation. In the following, we explain the configuration of protection connections and protected flows.

5.5.1 Configuration of Protection Connections. A protection connection is established by choosing registers on PTI and PTE to record the last sequence numbers SN_{last}^{PTI} and SN_{last}^{PTE} of a protection connection. The connection identifier is the PTE's index to access SN_{last}^{PTE}. On the PTI, a different index i may be chosen to access SN_{last}^{PTI}. Furthermore, the registers are initialized with zero. Moreover, the controller sets up a multicast group *m_grp* for each connection so that its traffic will be replicated in an efficient way to the two desired interfaces.

5.5.2 Configuration of Protected Flows. A protected flow is established by adding a new flow rule in the MAT ProtectedFlows of the PTI. It contains an appropriate flow descriptor and the parameters to call the action protect. Those are the index i associated with the corresponding protection connection, the CID needed at the PTE to identify the protection connection, the IP address of the PTI, the IP of the PTE, and the multicast group *m_grp*.

6 EVALUATION

In this section we evaluate the performance of the implemented mechanism on the Tofino Edgecore Wedge 100BF-32X. First, we compare packet processing times with and without P4-Protect. Then, we demonstrate that very high data rates can be achieved with and without P4-Protect on a 100 Gb/s interface. Finally, we show that P4-Protect can provide a transmission service with reduced jitter compared to the jitter of both protection tunnels.

6.1 Packet Processing Time

P4-Protect induces forwarding complexity. To evaluate its impact, we leverage P4 metadata to calculate the time a packet takes from the beginning of the ingress pipeline to the beginning of the egress pipeline. This is sufficient for a comparison as all work for P4-Protect is done in the ingress pipeline and all considered forwarding schemes utilizes the same egress pipeline. We compare three forwarding modes: a plain IP forwarding implementation (plain), P4-Protect for unprotected traffic (unprotected), and P4-Protect for protected traffic (protected).

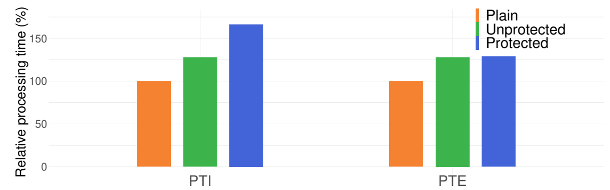


Figure 6: Ingress-to-egress packet processing time at PTI and PTE for three forwarding modes: plain, unprotected, and protected.

Figure 6 shows the ingress-to-egress packet processing time on both PTI and PTE for the three mentioned forwarding modes. The duration is given relative to the processing time for plain forwarding mode. We observe the lowest processing time at PTI and PTE for plain forwarding as it has the least complex pipeline. With P4-Protect, the processing time at both PTI and PTE is larger than with plain forwarding as the operations are more complex. At PTI, the processing time is even larger with protected forwarding (166%) than with unprotected forwarding (127%). At PTE, the processing times for protected and unprotected traffic are equal and 27% longer than with plain forwarding.

[1]Tofino is a high-performance chip which operates at 100 Gb/s so that only a limited set of operations can be performed for each packet, in particular in connection with register access.
[2]Repository: https://github.com/uni-tue-kn/p4-protect
[3]Repository: https://github.com/uni-tue-kn/p4-protect-tofino

In our implementations, we have used only a minimal IPv4 stack for all three forwarding modes. With a more comprehensive IPv4 stack, the relative overhead through P4-Protect is likely to be smaller.

6.2 TCP Goodput

We set up iperf3 connections between client/server pairs and measure their goodput. Each iperf3 connection consists of 15 parallel TCP flows. Two switches are bidirectionally connected via two 100 Gb/s interfaces. Four client/server pairs are connected to the switches via 100 Gb/s interfaces. Up to 4 clients download traffic from their servers via the trunk between the switches.

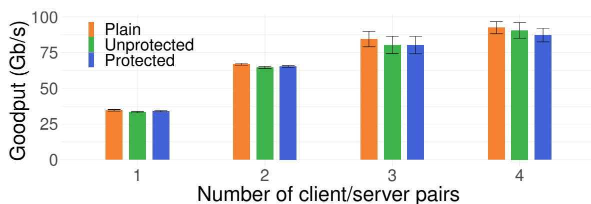


Figure 7: Impact of varying number of client/server pairs exchanging traffic with iperf3.

Figure 7 shows the overall goodput for a various number of client/server pairs, each transmitting traffic over a single TCP connection. The goodput is given for the forwarding modes plain, unprotected, and protected. We performed 20 runs per experiment and provide the 95% confidence interval.

A single, two, and, three TCP connections cannot generate sufficient traffic to fill the 100 Gb/s bottleneck link. However, with four TCP connections a goodput of around 90 Gb/s is achieved. This is less than 100% because of overhead due to Ethernet, IP, and TCP headers and due to the inability of TCP to efficiently utilize available capacity at high data rates. Most important is the observation that all three forwarding modes lead to almost identical goodput. The goodput for protected and unprotected forwarding is slightly lower than plain forwarding, which is apparently due to the operational overhead of P4-Protect.

6.3 Impact on Jitter

We examine the effect of 1+1 path protection on jitter. Two hosts are connected to two Tofino Edgecore Wedges 100BF-32X. The switches are connected with each other via two paths with intermediate Linux servers. Their interfaces are bridged and cause an artificial, adjustable, uniformly distributed jitter. We leverage the *tc* tool for this purpose [6]. All lines have a capacity of 100 Gb/s.

In our experiment, we send pings between the two hosts with and without P4-Protect. Figure 8 reports the average round trip time (RTT) deviation for the pings. Unprotected traffic suffers from all the jitter induced on a single path. Protected traffic suffers only from about half the jitter. This is because P4-Protect forwards the earliest received packet copy and minimizes packet delay occurred on both links.

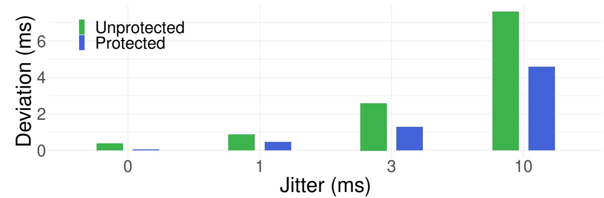


Figure 8: Impact of 1+1 protection on jitter.

7 CONCLUSION

In this paper we proposed P4-Protect for 1+1 path protection with P4. It may be utilized to protect traffic via two largely disjoint paths. We presented an implementation for the software switch bmv2 and the hardware switch Tofino Edgecore Wedge 100Bf-32X. The evaluation of P4-Protect on the hardware switch revealed that P4-Protect increases packet processing times only little, that high throughput can be achieved with P4-Protect, and that jitter is reduced by P4-Protect when traffic is carried over two path with similar delay but large jitter.

ACKNOWLEDGMENTS

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

REFERENCES

[1] A. Atlas et al. 2008. RFC5286: Basic Specification for IP Fast Reroute: Loop-Free Alternates .
[2] Ender Ayanoglu et al. 1993. Diversity coding for transparent self-healing and fault-tolerant communication networks. *IEEE ToC* 41(11) (1993).
[3] P. Bosshart et al. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM CCR* 44(3) (2014).
[4] Wolfgang Braun et al. 2017. Performance Comparison of Resilience Mechanisms for Stateless Multicast Using BIER. In *IFIP/IEEE*.
[5] Norman Finn, Pascal Thubert, Balazs Varga, and János Farkas. 2019. Deterministic Networking Architecture. RFC 8655. https://doi.org/10.17487/RFC8655
[6] Linux Foundation. 2019. *Linux Traffic Control*.
[7] A. Fumagalli et al. 2000. IP restoration vs. WDM protection: is there an optimal choice? *IEEE Network Magazine* 14(6) (2000).
[8] IEEE Computer Society. 2017. *Frame Replication and Elimination for Reliability*. Technical Report.
[9] ITU. 2006. ITU-T Recommendation G.803/Y.1342 (2006), Ethernet Protection Switching .
[10] ITU. 2010. ITU-T Recommendation G.7712/Y.1703 (2010), Internet protocol aspects – Operation, administration and maintenance.
[11] A. Karan et al. 2015. RFC7431: Multicast-Only Fast Reroute.
[12] James McCauley, Mingjie Zhao, Ethan J. Jackson, Barath Raghavan, Sylvia Ratnasamy, and Scott Shenker. 2016. The Deforestation of L2. In *Proceedings of the 2016 ACM SIGCOMM Conference*.
[13] Sèamas McGettrick et al. 2013. Ultra-fast 1+1 protection in 10 Gb/s symmetric Long Reach PON. In *IEEE ECOC)*.
[14] Daniel Merling et al. 2018. Efficient Data Plane Protection for SDN. IEEE (NetSoft).
[15] Christopher Metz. 2000. IP protection and restoration. *IEEE Internet Computing* 4(2) (2000).
[16] Mirzad Mohandespour et al. 2015. Multicast 1+1 protection: The case for simple network coding. In *IEEE ICNC*.
[17] Grazziela Niculescu et al. 2010. The Packet Delay in a MPLS Network Using "1+1 Protection. In *IEEE Advanced International Conference on Telecommunications*.
[18] Michal Przybylski, Bartosz Belter, and Artur Binczewski. 2005. Shall we worry about Packet Reordering? *Computational Methods in Science and Technology* 11.
[19] Jean Philippe Vasseur et al. 2004. *Network Recovery*. Morgan Kaufmann.
[20] Dongyun Zhou et al. 2000. Survivability in Optical Networks. *IEEE Network Magazine* 14(6) (2000).
[21] Harald Øverby et. al. 2012. Cost comparison of 1+1 path protection schemes: A case for coding. In *IEEE ICC*.

1.3 An Overview of Bit Index Explicit Replication (BIER)

ietfjournal.org

An Overview of Bit Index Explicit Replication (BIER)

19-24 minutes

Introduction

IP Multicast (IPMC) efficiently forwards one-to-many traffic and is leveraged for services like IPTV or multicast VPN (mVPN) [1]. In this article we explain the basic concept of traditional IPMC, describe its shortcomings, and present Bit Index Explicit Replication (BIER) as a solution.

An IPMC group may correspond to one specific IPTV channel. Packets destined to an IPMC group address are forwarded to all its members. Receivers leverage IGMP/MLD (Internet Group Management Protocol, RFC 3376/Multicast Listener Discovery, RFC 3810) to join an IPMC group. Within an IPMC domain, typical IPMC protocols use in-network traffic replication to ensure that at most a single copy of a packet traverses each link to reach multiple receivers. To that end, they establish per IPMC group one IPMC tree, possibly for each source, along which the traffic of that group is forwarded. The concept is shown in Figure 1. Examples for such protocols are PIM (Protocol Independent Multicast, RFC 7761), mLDP (Multicast Label Distribution Protocol), or RSVP-TE/P2MP (Resource Reservation Protocol – Traffic Engineering, RFC 3209, Point-to-Multipoint RFC 4875). The IPMC trees requires forwarding information in intermediate

hops that we denote as ‘state’ in the following.

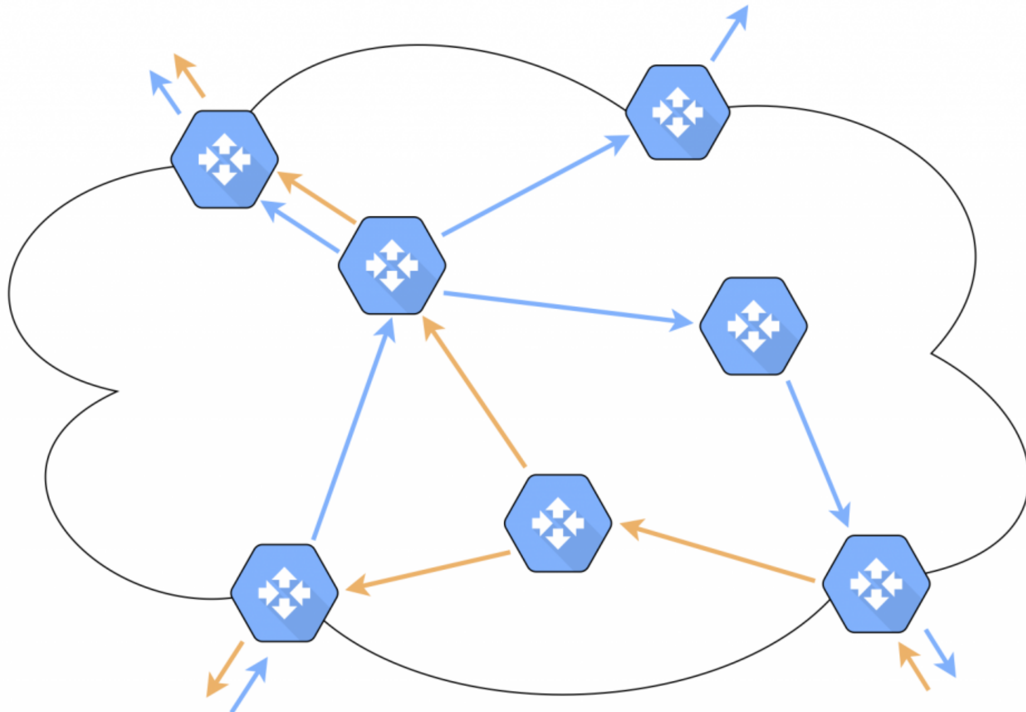


Figure 1: Two multicast trees.

Certain IPMC solutions for special use cases with static distribution trees – especially implementations of PIM – have proven to be useful and manageable. Nevertheless, traditional IPMC solutions suffer from limited scalability [1] [2]. Technologies to address these issues have been proposed but they cause further complexity and create new disadvantages.

BIER has been proposed by the IETF and is described in RFC 8279 [3]. The basic idea is to remove the IPMC-group-dependent state and the need for explicit-tree building from devices in the middle of the network to improve the scalability of the IPMC domain. This is achieved by adding a BIER header to IPMC packets. Within such a BIER domain, the packets are forwarded only according to this header.

Shortcomings of Traditional Multicast

Traditional IPMC solutions like PIM, mLDP, or RSVP-TE/P2MP

rely on per-group IPMC tree state. This tree state limits scalability in three ways.

P0. Devices have to store state per IPMC group.

P1. The IPMC protocol has to actively create, change, and tear-down the IPMC trees whenever IPMC groups start, change, or stop.

P2. In case of a topology change, the forwarding structure may need to change. Thus, the states of all IPMC groups possibly require adaptation. The time needed for that process scales with the number of IPMC groups.

Several additional technologies have been introduced to address these issues but they come with new disadvantages. Ingress replication is a tunnel-based approach that avoids additional state by utilizing unicast tunnels for building an IPMC tree at the expense of reduced forwarding efficiency. PMSI (Provider Multicast Service Interfaces, RFC 6513) leverages aggregated trees to carry the traffic of multiple IPMC applications, which causes significant signaling overhead. RSVP-TE/P2MP is a heavyweight approach to reduce convergence time issues for IPMC with pre-established backup tunnels. All those approaches have to be managed by operators making traditional IPMC more complex, expensive, less reliable, and overall challenging to deploy.

BIER proposes a replicating fabric technology which allows an operator to forward IPMC traffic efficiently without the need for explicit IPMC tree state in intermediate devices. In this section, we describe the concept of BIER, explain BIER's forwarding procedure in detail, and outline how it addresses the previously mentioned shortcomings of traditional IPMC.

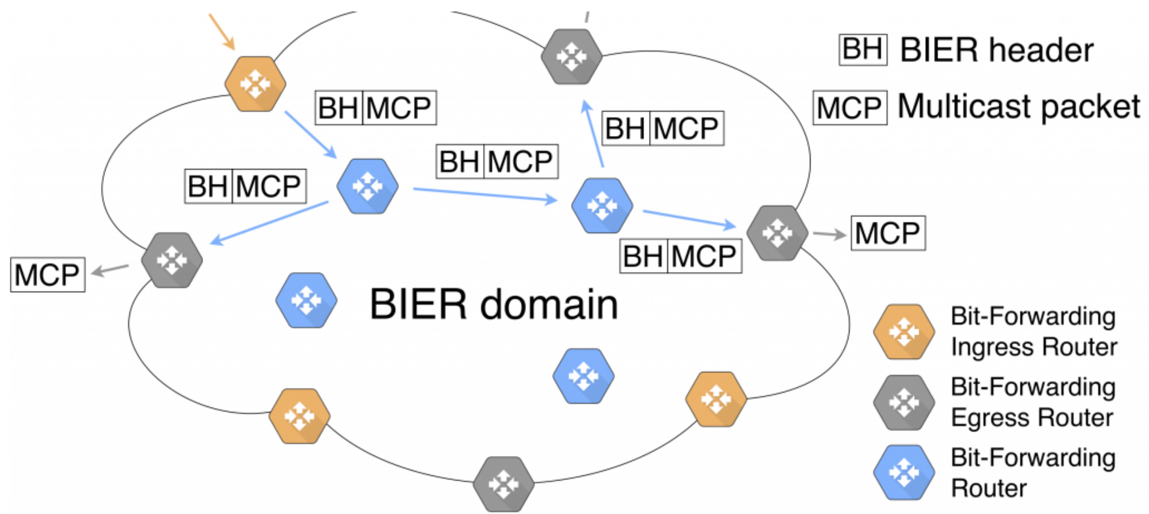


Figure 2: Packets enter the BIER domain via Bit-Forwarding Ingress Routers (BFIRs). They construct and push a BIER header onto the packet which holds information for BIER's forwarding procedure. At the Bit-Forwarding Egress Routers (BFERs), the BIER header is removed.

BIER Concept

The concept of BIER is illustrated in Figure 2. Traffic enters a BIER domain through a Bit-Forwarding Ingress Router (BFIR) and is replicated efficiently to potentially many Bit-Forwarding Egress Routers (BFERs). The BFIR adds a BIER header to the packets. This header contains information about the set of BFERs to which a copy of the packet is to be delivered. The BFERs remove the BIER header from the packets before they leave the BIER domain.

The BIER header is leveraged by all Bit-Forwarding Routers (BFRs) within the BIER domain to efficiently forward the traffic along a tree structure or even any acyclic graph that is determined from the underlay information, normally carried by the IGP (Interior Gateway Protocol). More specifically, the BIER header contains a bit string where each bit corresponds to a specific BFER. The BFIR sets that bit if the corresponding BFER should receive the packet.

A BFR relays and replicates BIER traffic based on that header information and its so-called Bit Index Forwarding Table (BIFT). The BIFT holds the next-hop information for every possible destination (BFER). Therefore, the size of the BIFT is independent of the number of IPMC groups. Real deployments may group the forwarding information for destinations that are reached via the same next-hop. This reduces the number of forwarding entries even further so that it scales with the number of a BFR's next-hops. The forwarding procedure ensures that a next-hop receives only a single copy of a packet even though the packet's BIER header indicates multiple destinations with that next-hop. To forward BIER traffic consistently, the BIFTs are commonly configured with shortest path entries towards the BFERs. BIER acquires this information from the IGP topology database of the underlying routing protocol, e.g. ISIS (Intermediate System to Intermediate System) or OSPF (Open Shortest Path First).

BIER Forwarding

In the following, we explain how traffic is forwarded with BIER along a shortest-path tree and illustrate it with an example. Figure 3 shows a network topology together with the shortest-path tree from Node 1 towards all destinations.

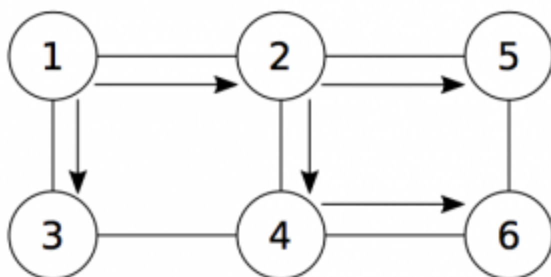


Figure 3: Example topology with the shortest-path forwarding tree for Node 1.

The BFERs are numbered and assigned to the bit positions in the bitstring of a BIER header. Thereby, counting starts with the least-significant bit of the bitstring. That means, the bitstring '000001' corresponds to Node 1 and '100000' corresponds to Node 6.

The BFR needs to ensure that all destinations receive a copy of the packet. To that end, the BFR forwards a copy to each next-hop that is on the path to at least one destination indicated in the BIER header. In our example, we assume that Node 1 receives a packet with a bitstring '100100' in the BIER header, i.e., the bits for Node 3 and Node 6 are activated. Therefore, Node 1 sends a copy of the packet to Node 3 and Node 2.

To prevent duplicates, a BFR clears all bits in the bitstring of a packet's BIER header that are not reached via the next-hop the packet is forwarded to. This ensures that there is only a single packet on the way towards each desired destination in spite of packet replication. In our example, Node 1 unsets the bit for Node 6 when forwarding the packet to Node 3 ('000100') and it unsets the bit for Node3 when forwarding the packet to Node 2 ('100000').

We explain how a BFR achieves the explained forwarding behavior in an efficient way using the bitstring of a packet's BIER header and its BIFT. The BIFT contains for every destination a so-called Forwarding Bit Mask (F-BM) and a next-hop. The F-BM is a bitmask whose bit positions correspond to the same BFERs as the bit positions in the bitstring of a BIER header. Activated bits in the F-BM indicate the BFERs that are reached via the specific next-hop. Therefore, all destinations reached via the same next-hop share the same F-BM. As an example, the BIFT of Node 1 is given in Table 1. For destination

Node 3, the next-hop is Node 3 and the corresponding F-BM indicates that only Node 3 is reached via Node 3. For the destinations Node 2, Node 4, Node 5, and Node 6, the next-hop is Node 2 and the F-BM indicates that all these nodes share the next-hop Node 2.

destination	F-BM	next-hop
1	-	-
2	111010	2
3	000100	3
4	111010	2
5	111010	2
6	111010	2

Table 1: BIFT of Node 1

To efficiently process a packet, the BFR creates an internal copy of the bitstring and performs the following algorithm until all bits of the internal copy of the bitstring are zero. The BFR finds a destination indicated in the bitstring of the internal copy of the bitstring. It looks up the F-BM for that destination in the BIFT and constructs a new BIER header using the bitstring of the packet ANDed with the F-BM. Then it sends a copy of the packet with the modified bitstring to the next-hop also indicated in the BIFT. Afterwards, the internal copy of the bitstring is modified by bitwise ANDing it with the complement of the F-BM. This action removes all destinations from the packet header that have been served by the last transmission of the packet.

BIER – A Scalable Multicast Approach

BIER overcomes the previously outlined problems of IPMC. It solves the problem of IPMC-group-dependent state within forwarding devices (P0) by moving this state to the BIER header.

In case of changing IPMC-groups (P1), only BFIRs require an update as they construct the BIER header that indicates the destinations of the packet. At last, the BIFT of every BFR holds forwarding entries for all BFERs in the network in a compact form. In case of a topology change (P2), only that information has to be updated instead of the tree state of potentially many IPMC groups, which takes a long time. As a result, the reconvergence time of BIER can be compared to IP unicast rather than to one of the traditional IPMC protocols.

By transferring the state from the forwarding devices to the header, the size of the header becomes a scalability issue as one bit is required for every BFER. With current router technology, 256 bits will be the most commonly used bitstring length because this is equivalent to the two IPv6 addresses in every IPv6 header. Longer bitstrings may be supported by future hardware. If there are more than 256 BFERs within the network, BIER supports the possibility of separating BFERs into subsets. The BIER header contains a field that identifies the subset that is addressed by a BIER packet. Thus, if an IPMC packet targets BFERs from different subsets, for each of these subsets, one copy of a packet has to be forwarded.

Use Cases

At the beginning of the BIER standardization journey, ten use cases were envisioned as technology drivers [1]. In this section we briefly describe the most prominent use cases, namely various multicast Layer 2/3 VPNs (L2/3VPNs), IPTV media streaming, data center virtualization services, and financial services. We outline problems that occur when these use cases are supported with traditional IPMC approaches and point out how BIER may be used to solve these problems.

Multicast VPN Services

Multicast within VPNs is used for news ticker, broadcast-TV applications or in general, content delivery networks (CDNs). For signaling in traditional multicast VPN (mVPN) services, PIM, mLDP, RSVP-TE/P2MP, or ingress replication is used. Each implementation offers a trade-off between state and flooding. The Multidirectional Inclusive PMSI (MI-PMSI) relies on flooding frames to all provider edge (PE) routers of the VPN, regardless of whether an IPMC receiver joined behind the PE routers. This results in a rather steady IPMC tree at the expense of flooding. In Selective PMSI (S-PMSI) only PE routers with joined receivers are part of the IPMC tree. S-PMSI reduces flooding with a more dynamic tree, requiring more state on the provider's core routers (P routers). Ingress replication causes the ingress PE router to send multiple copies of the same frame and forward it via unicast tunnels to the destinations. This poses a high replication burden on ingress routers and high bandwidth burden on paths.

Requiring IPMC-group-dependent state is a typical problem network operators are faced with (P0). With the introduction of BIER, this problem no longer exists.

IPTV Media Streaming

IPMC is leveraged for IPTV, or Internet video distribution in CDNs. Typical implementations like PIM, mLDP, or RSVP-TE/P2MP generate IPMC-group-dependent state as described in the previous use case. Additionally, such media streaming services may experience extensive subscription changes as every time a user switches a channel, the IPMC groups may have to be adapted. This may cause a high update frequency of

IPMC state.

BIER solves the problem of requiring IPMC-group-dependent state (P0). In particular changes of subscriptions can be managed by reconfiguring BFIRs instead of potentially many devices (P1) so that core routers are not affected.

Data Center Virtualization Services

Virtual eXtensible LAN (VXLAN, RFC 7348) interconnects L2 networks over an L3 infrastructure. It encapsulates L2 frames in UDP and adds a 24-bit ID so that 16 million virtual network instances (VNIs) can be differentiated. Each VNI is an isolated virtual network similar to a VLAN. That technology is used to isolate VLANs of multiple tenants in modern multi-tenant datacenters.

Typically, a tenant interconnects its virtual machines (VMs) over an L3 infrastructure using one or multiple VNIs to logically separate its own traffic and to isolate it from other tenants' traffic. If a VM is moved from one physical machine to another or even to another datacenter, there is no need to change its IP address as long as the VM remains in the same VNI.

IPMC can be leveraged to distribute broadcast, unknown, and multicast (BUM) traffic over the L3 infrastructure within a single VNI. One or even multiple IPMC groups are needed per tenant and, therefore, the number of IPMC groups may be very large. Thus, this use case faces again the IPMC state problem (P0), causing significant challenges for datacenter switches, data and forwarding planes, as well as for network operation and management. That problem may be solved by leveraging BIER instead of traditional IPMC protocols in the L3 underlay network.

Financial Services

IPMC is used to deliver real-time stock market data to subscribers. Such highly time-dependent data requires fast recalculation of paths in case of a topology change to satisfy latency requirements.

For traditional IPMC, a topology change requires a significant amount of time since potentially many IPMC trees have to be recomputed to restore connectivity and establish new shortest paths.

As BIER relies only on one IPMC-group-independent forwarding structure, its recomputation is significantly faster (P2).

Recent Working Group Achievements

The BIER working group developed BIER and provided several extensions, increasing its applicability and facilitating its deployment. We recap the results of the BIER working group below.

RFC 8279 [3] specifies the BIER architecture. Among others, it contains information about the BIER domain and its components, how the forwarding procedure works, and briefly explains the advantages of BIER compared to traditional IPMC solutions. RFC 8296 [4] defines the implementation of BIER encapsulation in MPLS and non-MPLS networks.

Signaling via PIM through the BIER domain, e.g. for subscriptions of receivers at a sender, is described in [9].

For operation in a real network, BIER devices need to share BIER-related information with each other. For example BFRs have to advertise their IDs, or bitstring lengths. BIER leverages link state routing protocols to perform this distribution. [5], [6] and [7] contain OSPF, ISIS and BGP extensions for this

purpose. The latter is supported by a document for a BGP link state extension for BIER [8].

Outlook

With the standardization of BIER, a new charter for the BIER working group [10] has been proposed. The main goal is to generate new experimental RFCs and to move existing experimental RFCs to the Standards Track.

The BIER working group has to define a transition mechanism for BIER. It should describe how BIER could be introduced in existing IPMC networks. This will facilitate the deployment of BIER.

The charter proposes documenting the applicability of BIER and its use cases. A draft for the application of BIER to multicast L3VPN and EVPN is required. Mechanisms for the signaling between ingress and egress routers and improving scalability are also mentioned. Furthermore, a document that clearly discusses the benefits of BIER for specific use cases is desired.

Operation, administration, and management of the BIER domain have to be described. The simplification of IPMC traffic management with BIER is a particular focus and for this purpose management APIs are required.

The BIER working group will continue the work on BIER-TE, an extension to BIER to support traffic engineering (TE). In software-defined networks (SDN), BIER may profit from a controller-based architecture. A controller may calculate the entries of the BIFTs and configure them in the BFRs. It may also instruct the BIFRs with appropriate BIER headers for encapsulation of traffic from specific IPMC groups.

Summary

BIER is a new, innovative mechanism for efficient forwarding and replication of IPMC traffic. It addresses scalability, operational, and performance issues of traditional IPMC solutions. While the latter require per-IPMC-group state and explicit-tree building in the forwarding devices, BIER encodes the destinations of an IPMC group within the packet's BIER header. The header is created by Bit-Forwarding Ingress Routers (BFIRs) when an IPMC packet enters the BIER domain. BIER scales very well as no IPMC-group-dependent information is required by forwarding nodes in the network core.

The collaboration in the BIER working group excels through participation of a large group of different vendors, operators, and researchers. Many companies have invested efforts in the standardization of BIER, which underlines its importance for future IPMC solutions. The spirit of the BIER working group is special even within the IETF. New ideas and use cases are always appreciated and discussed, and the community welcomes new members.

[1] Nagendra Kumar, Rajiv Asati, Mach Chen, Xiaohu Xu, Andrew Dolganow, Tony Przygienda, Arkadiy Gulko, Dom Robinson, Vishal Arya, and Caitlin Bestler. BIER Use Cases, January 2018.
[2] G. Shepherd, A. Dolganow, and A. Gulko. Bit Indexed Explicit Replication (BIER) Problem Statement. http://tools.ietf.org/html/draft-ietf-bier-problem-statement, April 2016.
[3] IJsbrand Wijnands, Eric C. Rosen, Andrew Dolganow, Tony Przygienda, and Sam Aldrin. Multicast Using Bit Index Explicit Replication (BIER). RFC 8279, November 2017.
[4] IJsbrand Wijnands, Eric C. Rosen, Andrew Dolganow, Jeff

Tantsura, Sam Aldrin, and Israel Meilik. Encapsulation for Bit Index Explicit Replication (BIER) in MPLS and Non-MPLS Networks. RFC 8296, January 2018.
[5] P. Psenak, N. Kumar, I. Wijnands, A. Dolganow, T. Przygienda, J. Zhang, and S. Aldrin. OSPF Extensions For BIER. https://datatracker.ietf.org/doc/draft-ietf-bier-ospf-bier-extensions/, October 2015.
[6] L. Ginsberg, A. Przygienda, S. Aldrin, and J. Zhang. BIER support via ISIS. https://datatracker.ietf.org/doc/draft-ietf-bier-isis-extensions/, October 2015.
[7] Xiaohu Xu, Mach Chen, Keyur Patel, IJsbrand Wijnands, and Tony Przygienda. BGP Extensions for BIER. Technical report, January 2018.
[8] Ran Chen, Zheng Zhang, Vengada Prasad Govindan, and IJsbrand Wijnands. BGP Link-State extensions for BIER. Technical report, February 2018.
[9] Hooman Bidgoli, Andrew Dolganow, Jayant Kotalwar, Fengman Xu, IJsbrand Wijnands, and mankamana prasad mishra. PIM Signaling Through BIER Core. Technical report, February 2018.
[10] Alia Atlas, Tony Przygienda, and Greg Shepherd. Charter for the BIER WG. https://datatracker.ietf.org/doc/charter-ietf-bier/, February 2018.

1.4 Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast

©2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast

Daniel Merling, Steffen Lindner, Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
{daniel.merling, steffen.lindner, menth}@uni-tuebingen.de

***Abstract*—IP multicast (IPMC) delivers one-to-many traffic along distribution trees. To that end, conventional IPMC requires state in forwarding devices for each IPMC group. This limits scalability of IPMC because forwarding state in core devices may be extensive and updates are necessary when IPMC groups or the topology change. The IETF introduced Bit Index Explicit Replication (BIER) for efficient transport of IPMC traffic. BIER leverages a BIER header and IPMC-group-independent forwarding tables for forwarding of IPMC packets in a BIER domain. However, legacy devices do not support BIER. In contrary, two SDN-based implementations for OpenFlow an P4 have been published recently. In this paper, we assess BIER forwarding which may be affected by network failures. So far there is no standardized procedure to handle such situations. Two concepts have been proposed. The first approach is based on Loop-Free Alternates. It reroutes traffic to suitable neighbors in the BIER domain to steer traffic around the failure. The second approach is a tunnel-based mechanism that tunnels BIER packets to appropriate downstream nodes within the BIER distribution tree. We explain and compare both approaches, and discuss their advantages and disadvantages.**

***Index Terms*—Software-Defined Networking, Bit Index Explicit Replication, Multicast, Resilience, Scalability**

I. Introduction

IP multicast (IPMC) is used for services like IPTV, commercial stock exchange, multicast VPN, content-delivery networks, or distribution of broadcast data. Figure 1 shows the concept of IPMC.

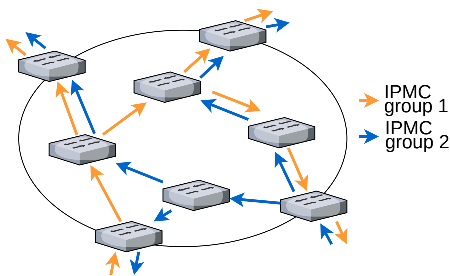


Figure 1: Two multicast distribution trees.

IPMC efficiently distributes one-to-many traffic by replicating packets and forwarding only one packet per link. Hosts join an IPMC group to receive the traffic addressed to that group. Forwarding devices maintain IPMC-group-dependent state to forward packets to the right neighbors. This decreases the scalability of IPMC for the following reasons. First, a large number of IMPC groups require a significant amount of forwarding state in core devices. Second, when subscribers of an IPMC group change, i.e., devices join or leave the group, the forwarding state needs to be updated. Third, when the topology changes or in case of a failure, the forwarding information base of possibly many devices has to be adapted.

The IETF presented BIER [1] as an efficient transport mechanism for IPMC traffic. BIER introduces a BIER domain, where only ingress routers maintain IPMC-group-dependent state. Ingress routers of the BIER domain encapsulate IPMC packets with a so-called BIER header which contains the destinations of the packet. Within the BIER domain, BIER packets are forwarded along distribution trees from the source to the destinations. Thereby only a single packet is transmitted per link. Finally, egress nodes remove the BIER header. Forwarding in the BIER domain is based on two components. First, the BIER header which contains a bit string that identifies receivers of a packet within the BIER domain. Second, the so-called Bit Index Forwarding Table (BIFT) which is the routing table of BIER devices. The entries of the BIFT are derived from information from the routing underlay, e.g., the Interior Gateway Protocol (IGP).

When a primary next-hop (NH) is unreachable due to a failure, an entire set of downstream destination nodes does not receive the traffic. When a failure is detected, IGP converges, new distribution trees are calculated, and the BIFTs are updated. This process requires a significant amount time. Therefore, BIER would benefit greatly from a fast protection mechanism that delivers traffic in the meantime. For unicast, several fast reroute (FRR) mechanisms [2] have been proposed which protect against the failure of single links or nodes until the forwarding information base is updated. FRR mechanisms use pre-computed backup entries to quickly reroute traffic when the primary NH is unreachable. No signaling between devices is necessary. Two FRR concepts for BIER have been proposed. First, LFA-based BIER-FRR [3] leverages a FRR mechanism called Loop-Free Alternates (LFAs) [4] that has been initially proposed for IP unicast. Failures are bypassed by forwarding traffic to alternative BIER NHs. Second, tunnel-based BIER-FRR tunnels traffic through the routing underlay, leveraging its FRR capabilities to steer traffic around the failure. We proposed this mechanism at the IETF [5].

However, legacy devices do not support BIER. On the contrary, the flexibility of SDN-based technologies have been leveraged recently to successfully implement BIER with OpenFlow [6] and in P4[1]. This allows the deployment of BIER

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

[1]https://github.com/uni-tue-kn/p4-bier

and facilitates the implementation of additional BIER-related features, e.g. BIER-FRR.

In this paper we review LFA-based and tunnel-based BIER-FRR. First, we propose changes to tunnel-based BIER-FRR to reduce the number of forwarding entries. Then, we point out major shortcomings of the LFA-based approach and present extensions to resolve the issues. Further, we compare both mechanisms by discussing their protection capabilities, and overhead in terms of header size and forwarding state.

The paper is structured as follows. Section II describes related work for conventional and SDN-based multicast, and BIER. We review BIER in Section III. Section IV gives a primer on LFAs. Then, in Section V we explain tunnel-based BIER-FRR. Afterwards, we describe LFA-based BIER-FRR in Section VI, and point out its shortcomings and propose extensions in Section VII. Finally, we compare and discuss both approaches in Section VIII. We conclude the paper in Section IX.

II. Related Work

In this section we first discuss related work for conventional and SDN-based multicast. Afterwards, we review related work for BIER.

A. Multicast

In [7] the authors provide an overview of the early development of multicast. The authors of [8] discuss the limited scalability of conventional IP multicast in terms of the number of forwarding entries. They propose an extension to the multicast routing protocol MOSPF to reduce the number of required forwarding entries. Li et al. [9] propose an architecture to partition the multicast address space to increase scalability of IP multicast in data center topologies.

B. SDN-Based Multicast

The surveys [10], [11] provide a detailed overview of SDN-based multicast. We discuss only some of the mentioned papers. The authors of [12] introduce software-defined multicast (SDM), an OpenFlow-based approach that aims at providing a well-managed multicast platform for over-the-top and overlay-based live streaming services. SDM is specifically engineered for the needs of P2P-based video stream delivery. They further develop their idea of SDM in [13] by adding support for fine-granular traffic engineering capabilities. Lin et al. [14] present a multicast model to construct so-called multi-group shared trees. By deploying distribution trees that cover multiple multicast groups simultaneously, the entire network is covered with a small number of trees.

C. Protection of SDN-Based Multicast

Kotani et al. [15] propose to leverage multiple simultaneously deployed multicast trees for protection. An ID in the packet header determines along which distribution tree a packet is forwarded. When a tree is affected by a failure, the controller reconfigures the senders to forward traffic on a backup tree. The authors of [16] follow a similar approach where they leverage primary and backup trees identified by a VLAN tag. When a switch detects a failure, it reroutes the packets on a working backup tree that contains all downstream nodes. This is accomplished by switching the VLAN tag in the packet header.

D. BIER Related Work

Giorgetti et al. [6], [17] provide an implementation for both, conventional IPMC and BIER forwarding in OpenFlow. They leverage MPLS headers to encode the BIER bit string, which limits the bit string length, and thereby the number of destinations, to a maximum of 20. However, a local BIER agent is required to run on the switches to support arbitrary destinations. BIER-TE [18] extends BIER with traffic engineering capabilities. BIER-TE leverages the same header format as BIER and supports explicit coding of a distribution tree in the BIER header. However, BIER and BIER-TE are not compatible. The authors of [19] present a P4-based implementation of BIER and BIER-TE and present different demo scenarios to show the feasibility and the advantages of BIER(-TE). The authors of [20] propose 1+1 protection for BIER-TE. Traffic for each IPMC group is forwarded on two disjoint distribution trees simultaneously. The trees share as few network components as possible to still deliver traffic when one tree is interrupted by a failure. However, the approach requires two forwarding planes, and in the failure free case twice the amount of network resources are occupied.

III. Bit Index Explicit Replication (BIER)

The following section reviews BIER [1]. First, we describe its concept, the structure of the Bit Index Forwarding Table (BIFT), the BIER forwarding procedure, and a forwarding example. Afterwards we explain a compact representation of the BIFT, and characteristics of the BIER topology.

A. BIER Concept

BIER is based on a layered architecture, consisting of routing underlay, BIER layer, and IPMC layer. Figure 2 illustrates the relation between these components.

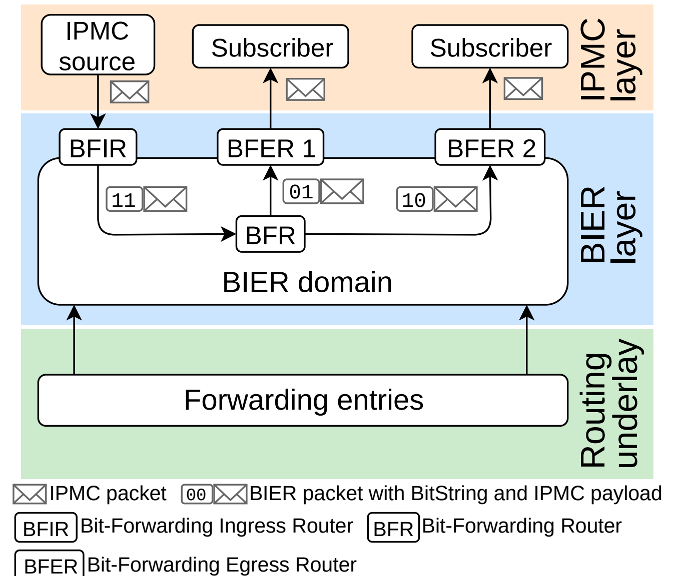


Figure 2: Layered architecture of BIER; it shows the relation between routing underlay, BIER layer, and IPMC layer.

The BIER layer serves as a point-to-multipoint tunnel for IPMC traffic through a BIER domain. The BIER domain consists of bit forwarding ingress routers (BFIRs), bit forwarding

routers (BFRs), and bit forwarding egress routers (BFERs). A BIER-capable device can be BFIR, BFR and BFER at the same time. When an IPMC packet enters the domain, the BFIR pushes a BIER header onto the IPMC packet. The BIER header identifies all receivers (BFERs) of the packet within the BIER domain. To that end, it contains a bit string which has to be at least as long as the number of BFERs in the BIER domain. In the following, 'BitString' refers to the bit string in the BIER header of the packet. Each BFER is assigned to a bit position in the BitString, starting with the least-significant bit. An activated bit means that the corresponding BFER must receive a copy of the BIER packet. BFRs forward BIER packets according to theor BitString along distribution trees to multiple BFERs.

Paths in the BIER domain are derived from the routing underlay, e.g., the IGP. As a consequence, BIER traffic follows the same paths as the corresponding unicast traffic from source to destination. At the domain boundary, BFERs remove the BIER header and pass the IPMC packet to the IPMC layer.

B. BIFT Structure

Table 1 shows the BIFT of BFR 1 from Figure 3. For each BFER, the BIFT contains one forwarding entry that consists of the primary NH and the so-called Forwarding Bit Mask (F-BM). The F-BM is a NH-specific bit string similar to the bit string in the BIER header. It indicates the BFERs with the same NH. In one particular F-BM, only bits of BFERs that are reached over the same NH are activated. During forwarding, BFRs use the F-BM to clear bits from the BitString.

C. BIER Forwarding

When a BFR receives a BIER packet, it stores its BitString to account to which BFERs the packet needs to be sent. We refer to that stored bit string by the term 'remaining bits'. The following procedure is repeated until the remaining bits do not contain any activated bits anymore.

The BFR determines the least-significant activated bit in the remaining bits. This bit indicates the BFER to be processed. Then, the BFR performs a looks up in the BIFT to get the NH and F-BM for that BFER. After a successful match, the BFR creates a copy of the received BIER packet. The BFR clears the BFERs from the BitString of the packet copy that have a different NH. To that end, the BFR performs a bitwise AND operation of the F-BM and the BitString of the packet copy. Then the BFR writes the result into the BitString of the packet copy. This procedure is called applying the F-BM. Thus, only bits that correspond to BFERs which share the same primary NH remain active in the BitString of the packet copy. Clearing other bits avoids duplicates at the receivers. Afterwards, the packet copy is forwarded to the NH. Finally, the BFERs, to which a packet has just been sent, are removed from the remaining bits. To that end, a bitwise AND operation of the bitwise complement of the F-BM and the remaining bits is performed.

D. BIER Forwarding Example

Figure 3 shows an example topology with four BFRs. Each BFR is in addition a BFIR and a BFER. Table 1 shows the BIFT of BFR 1.

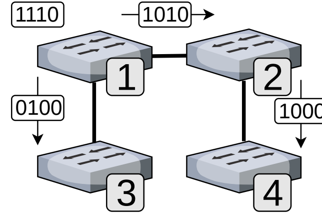


Figure 3: BIER topology and BitStrings of forwarded BIER packets.

BFER	NH	F-BM
1	-	-
2	2	1010
3	3	0100
4	2	1010

Table 1: BIFT of BFR 1.

BFR 1 receives a BIER packet with the BitString `1110`. The least-significant activated bit in the remainings bits identifies BFR 2. Therefore, BFR 1 creates a copy of the packet, applies the corresponding F-BM `1010`, and forwards the packet copy with the BitString `1010` to BFR 2. This sends a packet to BFER 2 and BFER 4. Afterwards, the bits of the F-BM are cleared from the remaining bits `0100`. The least-significant activated bit in the remaining bits corresponds to BFER 3. The F-BM is applied and a packet clone with the BitString `0100` is forwarded to the NH which is BFR 3. After clearing the F-BM from the remaining bits, processing stops because no active bits remain.

E. Compact BIFT

The number of entries of the BIFT scales with the number of BFERs. For improved scalability in terms of forwarding entries, the authors of [21] propose a compact representation of the BIFT that requires only one forwarding entry per neighbor. To that end, all entries with the same NH and F-BM are aggregated. As a result, all BFERs indicated in the F-BM share a single forwarding entry. During lookup, an entry is considered a match when at least one of the associated BFERs is a destination of the BIER packet. Table 2 shows the compact BIFT of BFR 1 from Figure 3.

BFERs	NH	F-BM
2, 4	2	1010
3	3	0100

Table 2: Compact BIFT of BFR 1.

F. Characteristics of the BIER Topology

In this paragraph we first discuss the impact of differences between the Layer 3 topology and BIER topology. Afterwards, we review how BIER devices detect whether BIER neighbors are still reachable.

1) Differences Between Layer 3 Topology and BIER Topology: In a Layer 3 topology some Layer 3 devices may not be BIER capable. Thus, the BIER topology may be different from the Layer 3 topology. Neighbors in the BIER topology are either connected directly to each other, or through at least one intermediate Layer 3 device that is no BIER device. BIER nodes receive information about their connection to their neighbors from the routing underlay. If two BIER neighbors are directly adjacent, they forward packets over Layer 2 to each other. If they are not directly adjacent, the BIER neighbors

leverage a Layer 3 tunnel to exchange packets. In both cases forwarding still follows the paths from the routing underlay.

2) Detection of Unreachable NHs: To quickly detect unreachable BIER neighbors, the authors of [22] propose bidirectional forwarding detection (BFD) [23] for BIER. When a BFD is established between two BIER nodes, they periodically exchange notifications to observe the reachability.

IV. Loop-Free Alternates

In this section we explain the concept of Loop-Free Alternates (LFAs) [4]. Afterwards, we review extensions for improved protection capabilities and loop detection.

A. Foundations of LFAs

LFAs implement a FRR mechanism for IP unicast traffic that prevents rerouting loops. Figure 4 shows the concept of LFAs.

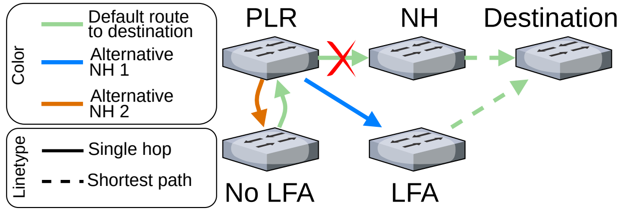


Figure 4: Concept of LFAs.

When a node cannot reach a primary NH, it acts as point of local repair (PLR), i.e., it leverages a pre-computed backup entry to reroute the packet via an alternative NH on a backup path towards the destination. Such neighbors are called LFAs and they have to be chosen in a way that rerouting loops are avoided. Some neighbors must not be chosen as LFAs because rerouting the packet would result in a forwarding loop.

LFAs have different properties for protection and loop avoidance. Some protect against link failures, others against node failures. Link-protecting LFAs (LP-LFAs) have a shortest path towards the destination that does not include the link between PLR and primary NH. Thus, LP-LFAs protect against the failure of the link between PLR and primary NH. The authors of [24] and [25] analyze the protection capabilities of LP-LFAs with a comprehensive set of topologies. They find that LP-LFAs protect only 70% of destinations against single link failures. Furthermore, LP-LFAs may cause loops when at least one node or multiple links fail instead of a single link only. To protect against the failure of the primary NH, node-protecting LFAs (NP-LFAs) have a shortest path to the destination that does not include the primary NH. In [24] the authors evaluate NP-LFAs in different scenarios on a large set of topologies. They show that NP-LFAs prevent loops for single link and single node failures, but they protect only 40% of destinations against single link failures.

B. Extensions for LFAs

In this paragraph we explain remote LFAs (rLFAs), topology independent LFAs (TI-LFAs), and explicit LFAs (eLFAs) to complement LFAs for increased protection capabilities. All three LFA variants support link and node protection. We indicate the protection mode with the prefix 'LP-' for link protection, and 'NP-' for node protection. Furthermore, we review a loop detection mechanism for LFAs. Figure 5 shows the concept of rLFAs, TI-LFAs, and eLFAs, which we explain in detail in the following.

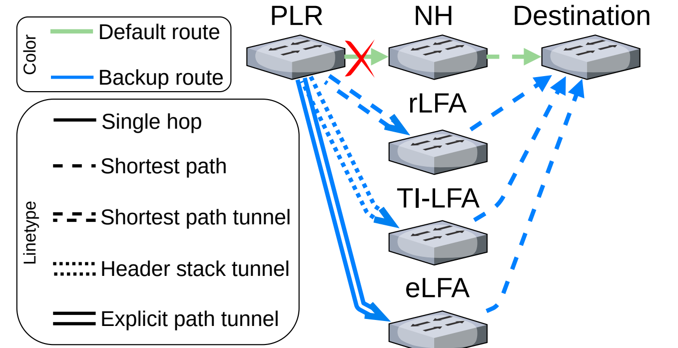


Figure 5: Concept of rLFAs, TI-LFAs, and eLFAs.

1) Remote LFAs (rLFAs): rLFAs [26] are remote nodes in the network. When the PLR cannot reach a primary NH, the packet is rerouted through a shortest path tunnel to the rLFA. From there, the packet is forwarded on a shortest path towards the destination. In [26] the authors prove that there is always a LP-rLFA to protect against a single link failure in unit-link-cost topologies. However, the authors of [25] find that this property does not hold for topologies with arbitrary link costs. NP-rLFAs cannot protect against all single link or single node failures.

2) Topology-Independent LFAs (TI-LFAs): TI-LFAs [27] are remote nodes in the network. When the primary NH is unreachable, the PLR leverages a header stack of IP headers to deviate traffic to the TI-LFA. The TI-LFA then sends the original packet on a shortest path towards the destination. As long as there is still a working shortest path to the destination, LP-TI-LFAs can protect against any single link failure, and NP-TI-LFAs against any single node failure.

3) Explicit LFAs (eLFAs): eLFAs [25] follow a similar concept as TI-LFAs. An eLFA is a remote node that serves as tunnel-end point when the PLR cannot reach the primary NH. The PLR reroutes the packet through an tunnel on an explicit path to the eLFA. The eLFA then forwards the packet on a shortest path to the destination. In contrast to TI-LFAs, eLFAs leverage additional forwarding entries for explicit paths to prevent an IP header stack. The authors of [25] evaluate eLFAs on a comprehensive set of different topologies. As long as the destinaton is still reachable, LP-eLFAs protect against any single link failure and NP-eLFAs protect against any single node failure.

4) Loop Detection: LFAs and all of its variants share the shortcoming that their deployment may cause forwarding loops [24], [25] in case of unprotected failures. In [24] the authors present a loop detection mechanism for LFAs. It is based on a bit string in the packet header where each forwarding device in the network is assigned a bit position. When a node needs to reroute a packet, it checks whether its own bit is activated. If this is not the case, the node activates the bit and reroutes the packet. However, if the bit is already activated, the packet

has been rerouted by the node before, and thus, the packet is dropped to prevent a loop. In [25] the authors describe loop detection for all LFA variants.

V. Tunnel-Based BIER-FRR

In this section we review tunnel-based BIER-FRR. We introduced this mechanism at the IETF [5]. First, we describe the concept, explain two modes of operation and an example. Then, we present changes to tunnel-based BIER-FRR for deployment with the compact BIFT. Finally, we discuss forwarding state.

A. Concept

When a BFR cannot forward a packet to a NH, the neighbor may still be reachable on a backup path. Tunnel-based BIER-FRR tunnels traffic through the routing underlay around the failure to BIER nodes downstream in the BIER distribution tree. A tunnel may be affected by the same failure but the routing underlay quickly restores connectivity with FRR mechanisms. With link protection, tunnel-based BIER-FRR tunnels the BIER packet to the NH. With node protection, BIER packets with adjusted BitStrings are tunneled to the next-next-hops (NNHs). Additionally, one BIER packet is tunneled to the NH to deliver a packet if only the link between PLR and NH failed.

Protection capabilities of tunnel-based BIER-FRR depend on the properties of the routing underlay. Tunnel-based BIER-FRR protects against any single component failure which can be handled by FRR mechanisms in the routing underlay. We describe the operation of tunnel-based BIER-FRR for link and node protection based on the normal BIFT.

1) Link Protection: Tunnel-based BIER-FRR with link protection does not require changes to the BIFT. When a primary NH is unreachable, the packet copy is tunneled to the NH instead of being forwarded on Layer 2. The routing underlay leverages IP-FRR to deliver the packet to the NH.

2) Node Protection: Tunnel-based BIER-FRR with node protection tunnels BIER packets to the NNHs. However, usually the NH adapts the BitString before the packet is forwarded to the NNH. Thus, before the packet is tunneled, the PLR performs modifications on the BitString that are usually done by the NH, i.e., applying the F-BM. To that end, backup entries in the BIFT are required which consist of a backup NH, and a backup F-BM. There are two categories of backup entries. First, for BFERs that are also NHs. In such backup entries, the NH is the backup NH and in the backup F-BM only the bit of the BFER is activated. This tunnels a packet to the NH in case only the link between PLR and NH failed. The second category of backup entries is for BFERs that are not NHs. For their entries, the backup NH is the NNH towards the BFER. The backup F-BM is the primary F-BM of the NH for the NNH.

When a primary NH is unreachable, the BFR performs three operations. First, the BFR applies the primary and the backup F-BM to the packet clone. The primary F-BM clears BFERs from the BitString that have a different NH. The backup F-BM clears BFERs from the BitString that have a different NNH. This leaves only bits of BFERs active that are activated in both, the primary and backup F-BM, i.e., all BFERs that have the same NH and the same NNH. Second, the packet copy is tunneled to the backup NH. Third, only bits that are active in both, the primary and backup F-BM are cleared from the remaining bits.

B. Forwarding Example

Figure 6 shows a BIER topology with a node failure where each BFR is also a BFIR and BFER. Table 3 displays the BIFT of BFR 1 with backup entries for node protection.

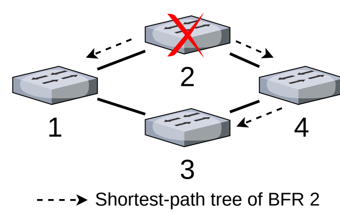


Figure 6: BIER topology with a node failure. The shortest-path tree of BFR 2 is shown to derive the backup F-BM of BFR 1 for BFER 4.

BFER	NH	F-BM
2	2	1010
	2	0010
3	3	0100
	3	0100
4	2	1010
	4	1100

Table 3: BIFT of BFR 1 with backup entries for node protection.

BFR 1 processes a packet with the BitString `1000`. The least-significant activated bit identifies BFER 4. However, the primary NH BFR 2 is unreachable. Thus, both, the primary F-BM `1010` and the backup F-BM `1100` are applied to the BitString of the packet copy. This leaves the BitString `1000` and the packet is tunneled to BFR 4 through the routing underlay. Bits that are activated in both, the primary and backup F-BM are cleared from the remaining bits which leaves `0000` and processing stops. The packet is eventually delivered by the routing underlay to BFR 4.

C. Compact BIFT

When the compact BIFT is used, tunnel-based BIER-FRR with link protection can be deployed as described in Section V-A1. Tunnel-based BIER-FRR with node protection requires two modifications. First, multiple backup entries are required for each primary forwarding entry. In the compact BIFT, each primary forwarding entry corresponds to one specific NH. For each NNH of the NH, one backup entry is required. The backup entries are calculated as described in Section V-A2. Second, when a BFR detects that a specific NH is unreachable, it matches incoming packets on the backup entries of the affected primary entry instead.

D. State Discussion

Tunnel-based BIER-FRR requires one backup entry for each primary entry. Therefore, in a topology with n BFERs the normal BIFT with backup entries contains $n + n$ forwarding entries. Deployment with the compact BIFT requires significantly fewer forwarding entries because the average

number of neighbors is significantly smaller than the number of destinations in a network. In a topology with an average node-degree of k, each node has k neighbors, and each NH has $k-1$ NHs on average. As a result the average number of forwarding entries per node is the sum of primary forwarding entries and backup entries $k + k \cdot (k-1)$.

VI. LFA-Based BIER-FRR

In this section we review LFA-based BIER-FRR [3]. We explain the concept, derivation of backup entries, and a forwarding example.

A. Concept

LFA-based BIER-FRR leverages backup entries in the BIFT to deviate traffic on backup paths when the primary path is interrupted. A backup entry consists of a backup NH, and a backup F-BM. When a primary NH is unreachable, further processing depends on the availability of a backup entry. If there is no backup entry, the bit of the BFER is cleared from the remaining bits and no packet is delivered to this particular BFER. Processing resumes with the next BFER. If there is a backup entry, further packet processing differs in three ways from regular BIER forwarding. First, the PLR applies the backup F-BM instead of the primary F-BM to the BitString of the packet clone. Second, the BIER packet is forwarded to the backup NH instead of the primary NH. Third, the bits of the backup F-BM instead of the primary F-BM are cleared from the remaining bits. Afterwards, the next BFER is processed.

B. Derivation of Backup Entries

We describe how we derive a backup entry consisting of a backup NH and a backup F-BM for a specific primary entry. First, we identify BIER neighbors that are LFAs as described in Section IV. LFA computation has to be performed on the BIER topology because Layer 3 LFAs may not be available on BIER layer due to topology differences. If no LFA is available, the primary forwarding entry remains without a backup entry. If there is an LFA L, it is selected as the backup NH. The activated bits in the backup F-BM are determined as follows. The bit that corresponds to an arbitrary BFER B is activated in the backup F-BM only if one of the two following conditions is fulfilled. First, L is an LFA to protect B. Second, L is the primary NH on the path to B. This aggregates all BFERs that are reached on a primary or backup path where L is the NH.

C. Forwarding Example

Figure 7 shows a BIER topology with a failed link between BFR 1 and 2. Each BFR is both a BFIR and a BFER. Table 4 contains the BIFT of BFR 1 with backup entries for link protection.

BFR 1 processes a BIER packet with the BitString `1110`. The least-significant activated bit identifies BFER 2. However, the primary NH BFR 2 is unreachable and there is no backup entry. Thus, the bit for BFER 2 is cleared from the remaining bits `1100` and no packet is sent. The next destination is BFER 3. Since the primary NH BFR 3 is reachable, the primary F-BM is applied and a packet clone with the BitString `0100` is forwarded to BFR 3. Clearing the F-BM from the remaining bits leaves only one bit activated `1000` which corresponds to BFER 4. However, the primary NH BFR 2 is unreachable. Thus, the backup F-BM is applied and a packet copy with the BitString `1000` is forwarded to the backup NH BFR 3. After the backup F-BM has been cleared from the remaining bits, no activated bits remain and processing stops. BFR 3 then forwards the packet to its destination BFR 4.

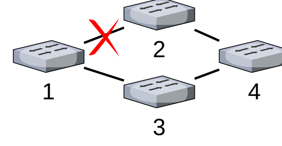


Figure 7: BIER topology with a link failure.

BFER	NH	F-BM
2	2 -	1010 -
3	3 -	0100 -
4	2 3	1010 1100

Table 4: BIFT of BFR 1 with backup entries for link protection.

VII. Extensions for LFA-Based BIER-FRR

In this section, we expose major shortcomings of LFA-based BIER-FRR in terms of matching order, coverage, and forwarding state, and propose solutions. In the end we discuss scalability in terms of forwarding entries.

A. Matching Order

In the previous example two packets are forwarded to BFR 3. This is caused by the order in which receivers of a packet are processed. The following scenario describes when more than one packet is forwarded to one specific NH P. First, a packet is forwarded to the primary NH P towards a set of BFERs. Second, another BFER that should receive the packet is processed but its primary NH is unreachable. However, P is the backup NH. Thus, a second packet is forwarded to P on a backup path. To avoid sending multiple packets over one link, it is necessary to first process forwarding entries whose primary NH is unreachable. Then, no additional packet is sent because the backup F-BM aggregates primary and backup paths that have the same NH.

B. Coverage

Depending on the topology, LFAs cannot protect against arbitrary single component failures. rLFAs protect against any single link failure on unit-link-cost topologies. TI-LFAs and eLFAs guarantee protection against any single component failure on arbitrary topologies. However, the deployment of each of the three LFA extensions requires some sort of IP or segment routing tunnel. Nevertheless, full protection is an important property and we suggest to augment LFA-based BIER-FRR with rLFAs, TI-LFAs, or eLFAs to increase the coverage. rLFAs, TI-LFAs, and eLFAs need to be BFRs. Therefore, computations have to be performed on the BIER topology because not all Layer 3 devices may be BIER devices.

C. Compact BIFT

We explain scalability issues of LFA-based BIER-FRR and propose a solution that requires changes to how backup entries are derived.

1) Problem Statement and Solution: LFA-based BIER-FRR has been described for the BIFT that contains one primary forwarding entry per BFER. In its proposed form LFA-based BIER-FRR is incompatible with the compact representation of the BIFT, which requires only one primary entry per neighbor. In the following we describe the necessary changes to use LFA-based BIER-FRR with the compact BIFT.

We propose to use a default BIFT that does not contain any backup entries and is used for forwarding in the failure-free case. In addition, we use failure-specific backup BIFTs. When a BFR detects that a specific neighbor is unreachable, it matches incoming packets on the backup BIFT that is associated with the unreachable NH. When the failure has been repaired or forwarding entries are updated, the BFR continues matching on the default BIFT.

2) Derivation of Backup BIFTs: We explain how the backup BIFT for a specific neighbor N is derived in two steps. First we fill the BIFT with entries and afterwards activate bits in specific F-BMs. We start with an empty backup BIFT. In the first step, for each neighbor that is not N, the corresponding primary entry from the default BIFT is added to the backup BIFT. In the second step, for each BFER B whose primary NH is N, LFAs are identified on the BIER topology. If an LFA is available, the bit that corresponds to B is activated in the F-BM of the BFR that is the LFA. If no LFA is available, B cannot be protected.

D. State Discussion

In a topology with n BFERs the normal BIFT contains n primary forwarding entries. LFA-based BIER-FRR requires n additional backup entries, which totals in $n + n$ forwarding entries. In contrast, the compact BIFT contains only one forwarding entry for each neighbor. Therefore, when the average node degree is k, the compact BIFT requires on average only k primary forwarding entries. On average each node has k backup BIFTs with on average $k - 1$ entries, which results in $k + k \cdot (k - 1)$ forwarding entries. Since the average node degree is significantly smaller than the number of destinations in a network, scalability of the compact BIFT is considerably better.

VIII. Comparison of LFA- and Tunnel-Based Protection for BIER

In this section we compare LFA-based and tunnel-based BIER-FRR. We point out similarities, and analyze protection capabilities and overhead with regard to header size and forwarding state. Afterwards, we discuss the impact of differences between Layer 3 topology and BIER topology.

A. Similarities

Both approaches implement FRR for BIER for resilient transport of IP multicast. Forwarding devices need to detect unreachable NHs, e.g. through a BFD. Both FRR mechanisms are based on pre-computed backup entries in addition to the primary forwarding entries. It is not necessary to change the structure of the BIFT. When the PLR cannot reach a primary NH, affected packets are rerouted according to the backup entries. Two modes of operation for link and node protection with different protection properties are available. For both, LFA- and tunnel-based BIER-FRR it is necessary to augment the forwarding procedure of BIER.

B. Protection Capabilities

We compare coverage properties and occurrence of loops.

1) Coverage: Tunnel-based BIER-FRR is able to protect traffic against arbitrary single component failures by design when the routing underlay provides full FRR coverage. As long as the destination is still reachable, an IP or segment routing tunnel is deployed to deliver the traffic to the unreachable NH or NNHs.

Protection of LFA-based BIER-FRR depends on the topology. The authors of [24] evaluate LP- and NP-LFAs on a comprehensive set of topologies. They find that LP-LFAs protect only 70% of destinations against single link failures and cause loops when nodes fail. NP-LFAs avoid loops when a node fails, but protect only 40% of destinations against single link failures. LP-rLFAs protect against any single link failure on unit link cost topologies. For any further guarantees TI-LFAs, or eLFAs have to be deployed. Both LFA extensions guarantee full protection for any single component failure in the network. However, augmenting LFA-based BIER-FRR with rLFAs, TI-LFAs, or eLFAs requires an additional header. TI-LFAs require an IP header stack, eLFAs require additional forwarding entries to implement backup paths.

2) Loops: Tunnel-based BIER-FRR cannot cause loops on the BIER layer because the packet is tunneled to the backup NH. When the packet is successfully delivered at the backup NH, BIER forwarding continues. If the tunnel is interrupted, the routing underlay is responsible for avoiding loops.

LFA-based BIER-FRR cannot guarantee to avoid loops because depending on the failure scenario and the mode of operation, all LFA variants can cause loops [24], [25]. With link protection, traffic may loop if at least one node or multiple links fail. With node protection, loops are prevented as long as not multiple components fails. In Section IV-B4 we review a loop detection mechanism for LFAs and all variants to prevent loops in any failure scenario. However, this mechanism significantly increases operational complexity and modifications to the packet header are necessary.

C. Overhead

We compare both protection approaches according to packet header size and required forwarding state.

1) Header Size: Tunnel-based BIER-FRR requires tunneling to protect traffic against failures. This adds an additional header to the packet. When the tunnel is interrupted and the routing underlay leverages a tunnel-based FRR protection mechanism for unicast, e.g. TI-. or eLFAs, an additional header is added to the packet. The basic form of the LFA-based BIER-FRR approach does not require tunneling. However, rLFAs, TI-LFAs, or eLFAs increase the protection capabilities of LFAs to an appropriate level but require at least one additional IP

header. More header reduce the throughput and the Maximum Transmission Unit (MTU) has to be decreased at domain boundaries. The LFA-based approach requires a loop detection mechanism to prevent loops. Such a mechanism is available, however it increases packet header size even further.

2) Forwarding State: Both BIER-FRR approaches require the same amount of forwarding state. In a topology with n BFERs and an average node degree of k, the regular BIFT contains $n + n$ forwarding entries while the compact BIFT requires on average only $k + k \cdot (k - 1)$ entries. Since k is significantly smaller than n, deployment with the compact BIFT provides better scalability.

D. Influence of the BIER Topology

When some network nodes in a Layer 3 network do not support BIER, Layer 3 LFAs may disappear on the BIER layer. Thus, coverage of LFA-based BIER-FRR depends on the BIER topology. When regular LFAs have low coverage, LFA-based BIER-FRR needs to be complemented with rLFAs, TI-LFAs, or eLFAs. Backup paths may become longer in a sparse BIER topology because LFAs may be reachable only through a long Layer 3 tunnel. Tunnel-based BIER-FRR leverages tunnels through the routing underlay to the BIER NH or BIER NNHs for protection. Thus, tunnel-based BIER-FRR is not affected in a negative way by a BIER topology that is different from the Layer 3 topology.

IX. Conclusion

In this paper we compared LFA-based and tunnel-based BIER-FRR for resilient and scalable transport of IP multicast. Our discussion showed shortcomings of the LFA-based approach. Sometimes multiple packets are sent over one link, not all single link or single node failures can be protected, and in some scenarios backup traffic may loop. We propose extensions to overcome those shortcomings so that the capabilities of LFA-based and tunnel-based BIER-FRR mechanisms are equal. Differences remain in backup path length when the BIER topology is different from the Layer 3 topology, and in the need for additional headers.

References

[1] I. Wijnands, E. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, *RFC 8279: Multicast Using Bit Index Explicit Replication (BIER)*, https://datatracker.ietf.org/doc/rfc8279/, Nov. 2017.

[2] J. Papán, P. Segeč, M. Moravčík, M. Kontšek, L. Mikuš, and J. Uramová, "Overview of IP Fast Reroute solutions," in *ICETA*, 2019.

[3] I. Wijnands, G. J. Shepherd, C. J. Martin, and R. Asati, *Per-Prefix LFA FRR with Bit Indexed Explicit Replication*, https://patents.google.com/patent/US20180278470A1/en, Sep. 2018.

[4] G. Rétvári, J. Tapolcai, G. Enyedi, and A. Császár, "IP fast ReRoute: Loop Free Alternates revisited," in *IEEE INFOCOM*, 2011.

[5] D. Merling and M. Menth, *BIER Fast Reroute*, https://tools.ietf.org/html/draft-merling-bier-frr-00, Mar. 2019.

[6] A. Giorgetti, A. Sgambelluri, F. Paolucci, P. Castoldi, and F. Cugini, "First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting," in *IEEE EuCNC*, 2017.

[7] K. C. Almeroth, "The Evolution of Multicast: From the MBone to Interdomain Multicast to Internet2 Deployment," *IEEE Network*, vol. 14, 2000.

[8] B. Zhang and H. T. Mouftah, "Forwarding State Scalability for Multicast Provisioning in IP Networks," *IEEE ComMag*, vol. 41, 2003.

[9] X. Li and M. J. Freedman, "Scaling IP Multicast on Datacenter Topologies," in *ACM CoNEXT*, 2013.

[10] S. Islam, N. Muslim, and J. W. Atwood, "A Survey on Multicasting in Software-Defined Networking," *IEEE Comst*, vol. 20, 2018.

[11] Z. AlSaeed, I. Ahmad, and I. Hussain, "Multicasting in Software Defined Networks: A Comprehensive Survey," *JNCA*, vol. 104, 2018.

[12] J. Rückert *et al.*, "Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks," *JNSM*, vol. 23, 2015.

[13] J. Rückert, J. Blendin, and D. Hausheer, "Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm," *IEEE TNSM*, vol. 13, 2016.

[14] Y.-D. Lin, Y.-C. Lai, H.-Y. Teng, C.-C. Liao, and Y.-C. Kao, "Scalable Multicasting with Multiple Shared Trees in Software Defined Networking," *JNCA*, vol. 78, 2017.

[15] D. Kotani, K. Suzuki, and H. Shimonishi, "A Multicast Tree Management Method Supporting Fast Failure Recovery and Dynamic Group Membership Changes in OpenFlow Networks," *JIP*, vol. 24, 2016.

[16] T. Pfeiffenberger, J. L. Du, P. B. Arruda, and A. Anzaloni, "Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow," in *IFIP NTMS*, 2015.

[17] A. Giorgetti, A. Sgambelluri, F. Paolucci, N. Sambo, P. Castoldi, and F. Cugini, "Bit Index Explicit Replication (BIER) Multicasting in Transport Networks," in *ONDM*, 2017.

[18] T. Eckert, G. Cauchie, W. Braun, and M. Menth, *Traffic Engineering for Bit Index Explicit Replication BIER-TE*, http://tools.ietf.org/html/draft-eckert-bier-te-arch, Nov. 2017.

[19] W. Braun, J. Hartmann, and M. Menth, "Demo: Scalable and Reliable Software-Defined Multicast with BIER and P4," in *IFIP/IEEE IM*, 2017.

[20] W. Braun, M. Albert, T. Eckert, and M. Menth, "Performance Comparison of Resilience Mechanisms for Stateless Multicast using BIER," in *IFIP/IEEE IM*, 2017.

[21] Z. Zhang and A. Baban, *Bit Index Explicit Replication (BIER) Forwarding for Network Device Components*, https://patents.google.com/patent/US20160191372, Dec. 2014.

[22] Q. Xiong, G. Mirsky, F. Hu, and C. Liu, *BIER BFD*, https://datatracker.ietf.org/doc/draft-hu-bier-bfd/, Oct. 2017.

[23] D. Katz and D. Ward, *Bidirectional Forwarding Detection (BFD)*, https://datatracker.ietf.org/doc/rfc5880/, Jul. 2004.

[24] W. Braun and M. Menth, "Loop-Free Alternates with Loop Detection for Fast Reroute in Software-Defined Carrier and Data Center Networks," *JNSM*, vol. 24, 2016.

[25] D. Merling, W. Braun, and M. Menth, "Efficient Data Plane Protection for SDN," in *IEEE NetSoft*, 2018.

[26] L. Csikor and G. Rétvári, "IP fast reroute with remote Loop-Free Alternates: The unit link cost case," in *ICUMT*, 2012.

[27] P. Francois, C. Filsfils, A. Bashandy, B. Decraene, and S. Litkowski, *Topology Independent Fast Reroute using Segment Routing*, https://tools.ietf.org/html/draft-francois-rtgwg-segment-routing-ti-lfa-00, Aug. 2015.

1.5 P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast

P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast

Daniel Merling*, Steffen Lindner, Michael Menth

University of Tuebingen, Department of Computer Science, Chair of Communication Networks, Tuebingen, Germany

Abstract

Bit Indexed Explicit Replication (BIER) is a novel IP multicast (IPMC) forwarding paradigm proposed by the IETF. It offers a transport layer for other IPMC traffic, keeps core routers unaware of IPMC groups, and utilizes a routing underlay, e.g., an IP network, for its forwarding decisions. With BIER, core networks do not require dynamic signaling and support a large number of IPMC groups with large churn rates. The contribution of this work is threefold. First, we propose a novel fast reroute (FRR) mechanism for BIER (BIER-FRR) so that IPMC traffic can be rerouted as soon as the routing underlay is able to carry traffic again after a failure. In particular, BIER-FRR enables BIER to profit from FRR mechanisms in the routing underlay. Second, we describe a prototype for BIER and BIER-FRR within an IP network with IP fast reroute (IP-FRR). It is based on the programmable data plane technology P4. Third, we propose a controller hierarchy with local controllers for local tasks, in particular to enable IP-FRR and BIER-FRR. The prototype demonstrates that BIER-FRR reduces the protection time for BIER traffic to the protection time for unicast traffic in the routing underlay.

Keywords: Software-Defined Networking, P4, Bit Index Explicit Replication, Multicast, Resilience, Scalability

---

©2020. This manuscript version is made available under the CC-BY-NC-ND 4.0 license `http://creativecommons.org/licenses/by-nc-nd/4.0/`

`https://doi.org/10.1016/j.jnca.2020.102764`

*Corresponding author

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/2. The authors alone are responsible for the content of the paper.

Email addresses: `daniel.merling@uni-tuebingen.de` (Daniel Merling), `steffen.lindner@uni-tuebingen.de` (Steffen Lindner), `menth@uni-tuebingen.de` (Michael Menth)

1. Introduction

IP multicast (IPMC) is leveraged for many services like IPTV, multicast VPN, or the distribution of financial or broadcast data. It efficiently forwards one-to-many traffic on tree-like structures to all desired destinations by sending at most one packet copy per link in the distribution tree. IPMC is organized into sets of receivers, so-called IPMC groups. Hosts subscribe to IPMC groups to receive the traffic which is addressed to that group. Traditional IPMC methods require per-IPMC-group state within core routers to forward the packets to the right next-hops (NHs). This raises three scalability issues. First, the number of IPMC groups may be large which require lots of space in forwarding tables. Second, core routers are involved in the establishment, removal, and in the change of an IPMC group. This requires significant signaling in the core network every time subscribers change because many nodes possibly need to update their forwarding information base, which imposes heavy load on core when churn rates are large. Third, when the topology changes or in case of a failure, the forwarding of any IPMC group possibly requires fast update, which is demanding in a critical network situation. IPMC features are available in most off-the-shelf hardware, but the features are turned off by administrators due to complexity and limited scalability.

The Internet Engineering Task Force (IETF) developed Bit Index Explicit Replication (BIER) [1, 2] as a solution to those problems. BIER features a domain concept. Only ingress and egress routers of a BIER domain participate in signalling. They encapsulate IPMC packets with a BIER header containing a bit string that indicates the receivers of the IPMC group within the BIER domain. Based on that bit string the packets are forwarded through the BIER domain without requiring per-IPMC-group state in core routers.

BIER leverages the so-called bit indexed forwarding table (BIFT) for forwarding decisions. Its entries are derived from paths induced by the interior gateway protocol (IGP) which is used for unicast routing. In the following we refer to that routing protocol with the term 'routing underlay'. Therefore, BIER traffic follows the same paths as the unicast traffic carried by the routing underlay. So far, BIER lacks any protection capabilities. In case of a link or node failure, the BIFT entries need to be changed so that BIER traffic is carried around failed elements towards receivers. However, the BIFTs can be updated only after the routing underlay has updated its forwarding information base and based on the new paths, BIER forwarding entries are recomputed. This takes a significant amount of time. In the meantime, packets are dropped. When a multicast packet is dropped, all downstream subscribers cannot receive the packet. Regular IP forwarding is affected as well by failures, but for unicast traffic, fast reroute (FRR) [3] mechanisms have been proposed to reroute affected packets on backup paths until the primary forwarding entries are updated. IP-FRR leverages pre-computed backup actions for fast recovery in case of a failure without the need for signaling. However, IP-FRR is not a suitable protection method for multicast traffic because it does not consider the tree-like forwarding structures along which IPMC packets are distributed.

In this work, we introduce BIER-FRR. It has two different operation modes to protect either only against link failures or also against node failures. We recently proposed this mechanism in the BIER working group of the IETF [4]. BIER has been suggested as a novel transport mechanism for IPMC. However, it cannot be configured yet on standard hardware. New, programmable data plane technologies allow the definition of new packet headers and forwarding behavior, and offer themselves for the implementation of prototypes. In [5], we presented an early version of a P4-based prototype for BIER. It was based on the P_{14} specification of P4 [6] and required a few workarounds because at that time some P4 features were not available on our target, the software switch BMv2. Moreover, there was no protection method available for BIER. We now provide the description of a completely reworked prototype on the base of the P_{16} specification of P4 [7]. The new prototype implements IP forwarding, a simple form of IP-FRR, BIER forwarding, and BIER-FRR. It comprises a controller hierarchy with local controllers that enables FRR techniques with P4. We argue that local controllers are needed for protection and helpful for local tasks in general. The evaluation of the prototype shows that BIER traffic is not longer affected by network failures than unicast traffic when BIER-FRR is enabled. Thus, the contribution of this paper is threefold: (1) a concept for BIER-FRR, (2) an implementation of BIER and BIER-FRR with P4, and (3) a controller hierarchy with local controllers to support FRR techniques in P4. Finally, the P4-based prototype demonstrates the usefulness of BIER-FRR. The source code of our prototype with the fully working data and control plane is publicly available on GitHub.

The remainder of this paper is structured as follows. Section 2 reviews basics of multicast. Section 3 contains fundamentals about IP-FRR, explains why it is insufficient to protect multicast traffic, and examines related work. Section 4 discusses related work for both legacy- and SDN-based multicast. Section 5 gives a primer on BIER. Section 6 explains the resilience problem of BIER and introduces BIER-FRR. In Section 7 we summarize necessary basics of P4 needed for the understanding of the BIER prototype. Section 8 describes the P4-based prototype implementation of IP, IP-FRR, BIER, and BIER-FRR. The prototype is used to demonstrate the usefulness of BIER-FRR in Section 9. Finally, Section 10 summarizes this paper and discusses further research issues.

2. Technological Background for IP Multicast

This section gives a primer on IP multicast (IPMC) for readers that are not familiar with IPMC. IPMC supports one or more sources to efficiently communicate with a set of receivers. The set of receivers is called an IPMC group and is identified by an IP address from the Class D address space (224.0.0.0 – 239.255.255.255). Devices join or leave an IPMC group leveraging the Internet Group Management Protocol (IGMP) [8].

IPMC packets are delivered over group-specific distribution trees which are computed and maintained by IPMC-capable routers. In the simplest form, source-specific multicast trees based on the shortest path principle are computed

and installed in the routers. The notation (S, G) identifies such a shortest path tree for the source S and the group G.

IPMC also supports the use of shared trees that can be used by multiple senders to send traffic to a multicast group. The shared tree has a single root node called rendezvous point (RP). The sources send the multicast traffic to the RP which then distributes the traffic over a shared tree. In the literature, shared trees are denoted by $(*, G)$.

Protocol-independent multicast (PIM) leverages unicast routing information to perform multicast forwarding. PIM cooperates with various unicast routing protocols such as OSPF or BGP and supports both source-specific and shared multicast distribution trees.

Pragmatic General Multicast (PGM) [9] reduces packet loss for multicast traffic. It enables receivers to detect lost or out-of-order packets and supports retransmission requests similar to TCP.

3. IP Fast Reroute

In this section we give a primer on IP fast reroute (IP-FRR). First, we explain fundamentals of IP-FRR and describe Loop-Free Alternates. Then, we discuss related work.

3.1. Fundamentals of IP-FRR

When a link or a node fails, devices may not be able to forward packets to their NHs. As soon as a failure is detected in an IP network, the changed topology is signaled through the network, new working paths are calculated, and the forwarding tables of all devices are consistently updated. This process is called reconvergence and may take up to several seconds. In the meantime, packets are dropped in case of wrong or missing forwarding entries. IP-FRR [3] protects IP unicast traffic against the failure of single links and nodes while reconvergence is ongoing. It is based on pre-computed backup actions to quickly reroute affected packets. Figure 1 shows an example for Loop-Free Alternates (LFAs) [10] which is the most popular IP-FRR mechanism. When a node's

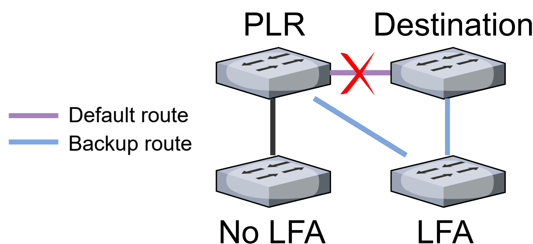


Figure 1: A PLR reroutes a packet to a backup path when the NH on the primary path is unreachable.

(primary) NH becomes unreachable, the node detects that failure after some

time and reroutes the traffic locally over a backup path. Therefore, the node is also called point of local repair (PLR). To reroute packets in a timely manner, nodes store a backup NH in addition to the primary NH for each destination. When the PLR detects that the primary NH is unrechable, e.g., by loss-of-light detection, loss-of-carrier detection, a bidirectional forwarding detection[1] (BFD) [11], or any other suitable mechanism, it forwards the packet to its backup NH instead. That backup NH is called Loop-Free Alternate (LFA) and it has to be chosen such that rerouted traffic does not loop. However, some destinations remain unprotected because there is not always an alternative hop that has a shortest path towards the destination which avoids the failed element. The set of protected destinations is also called coverage. The limited coverage of LFAs has been evaluated in various studies [12, 13].

3.2. Related Work

The two surveys [14, 15] give an overview of several IP-FRR mechanisms. We discuss only some of the papers. Equal-cost multi-path (ECMP) can be used as a very basic FRR mechanism. When a PLR has at least two paths with equal cost towards a destination, it quickly deviates traffic to the other path when the primary NH is unreachable. However, this works only if two equal-cost paths are available under normal conditions, which is mostly not the case. Not-via addresses [16, 13] tunnel packets to the downstream next-next-hop (NNH) if the NH is unreachable. To that end, the NNH is assigned a unique address and an explicit backup path is constructed which does not include the failed component. Loop-Free Alternates (LFAs) [10] forward packets to alternative NHs if the primary NH is unreachable. Those alternative NHs have to be chosen in a way that they have a working shortest path to the destination that avoids rerouting loops. As such alternative neighbors do not exist for all PLRs and destinations, the LFA mechanism cannot protect against all single link and node failure. Remote LFAs [17] (rLFAs) extend the protection capabilities of LFAs by sending affected packets through shortest path tunnels to nodes that still reach the destination on a working shortest path. rLFAs protect against any single link failure in unit link cost networks. However, they achieve only partial coverage in case of node failures or non-unit link costs. An analysis can be found in [12].

4. Related Work

We review work in the context of SDN-based multicast. Most traditional multicast approaches were implemented with OpenFlow. Some works considered protection mechanisms. A few studies improve the efficiency of multicast forwarding with SDN. Only a single work implements BIER without protection using OpenFlow, but the implementation itself requires dynamic forwarding state, which runs contrary to the intention of BIER.

[1]When a BFD is established between two nodes, they periodically exchanges keep-alive notifications.

4.1. Multicast Implementations with OpenFlow

The surveys [18, 19] provide an extensive overview of multicast implementations for SDN. They discuss the history of traditional multicast and present multiple aspects for SDN-based multicast, e.g., multicast tree planning and management, multicast routing and reliability, etc. In the following we briefly discuss some exemplary works that implement multicast for SDN. More details can be found in the surveys or the original papers.

Most related works with regard to SDN-based multicast implement explicit flow-specific multicast trees. Most authors propose to compute traffic-engineered multicast trees in the controller using advanced algorithms and leverage SDN as tool to implement the multicast path layout. The following works provide implementations in OpenFlow to show the feasibility of their approaches. Dynamic Software-Defined Multicast (DynSDM) [20, 21] leverages multiple trees to load-balance multicast traffic and efficiently handle group subscription changes. Modified Steiner tree problems are considered in [22, 23] to minimize the total cost of edges and the number of branch nodes, or to additionally minimize the source-destination delay [24]. In [25], the authors compute and install traffic-engineered shared multicast trees using OpenFlow. In [26] and [27], traffic-engineered Steiner trees are computed which minimize the number of edges of the tree and provide optimized locations for multicast sources in the network. The Avalanche Routing Algorithm (AvRA) [28] considers topological properties of data center networks to optimize utilization of network links. Dual-Structure Multicast (DuSM) [29] improves scalability and link utilization for SDN-based data center networks by deploying different forwarding approaches for high-bandwidth and low-bandwidth flows. In [30], Steiner trees are leveraged to compute a multicast path layout including certain recovery nodes which are used for reliable multicast transmission such as PGM. In [31], the authors evaluate different node-redundant multicast tree algorithms in an SDN context. They evaluate the number of forwarding rules required for each mechanism and study the effects of node failures. The authors of [32] reduce the number of forwarding entries in OpenFlow switches for multicast. They propose to use address translation from the multicast address to the receiver's unicast address on the last branching node of the multicast tree. This allows to omit multicast-specific forwarding entries in leaf switches.

4.2. Multicast Protection with OpenFlow

Kotani et al. [33] suggest to utilize primary and backup multicast trees for SDN networks. Multicast packets carry an ID to identify the distribution tree over which they are forwarded. In case of a failure, the controller chooses an appropriate backup multicast tree and reconfigures senders accordingly. This mechanism differs in two ways from BIER-FRR. First, the controller has to be notified, which is not suitable for fast recovery. Second, it requires significant signalling effort in response to a failure.

The authors of [34] propose a FRR method for multicast in OpenFlow networks. Multicast traffic is forwarded along a default distribution tree. If a

downstream neighbor is unreachable, traffic is switched to a backup distribution tree that contains all downstream nodes of the unreachable default subtree. The backup distribution tree must not contain the unreachable neighbor as forwarding node. VLAN tags are used to indicate the trees over which multicast packets should be sent. This mechanism differs from BIER-FRR in a way that it requires a significant amount of additional dynamic forwarding state to configure the backup trees.

4.3. Improved Multicast Forwarding for SDN Switches

Some contributions improve the efficiency of devices to enable hardware-based multicast forwarding. The work in [35] organizes forwarding entries of a switch based on prime numbers and the Chinese remainder theorem. It reduces the internal forwarding state and allows for more efficient hardware implementations. Reed et al. provide stateless multicast switching in SDN-based systems using Bloom filters in [36] and implement their approach for TCAM-based switches. The authors compare their approach with existing layer-2 forwarding and show that their method leads to significantly lower TCAM utilization.

4.4. SDN Support for BIER

The authors of [37, 38] implement two SDN-based multicast approaches using (1) explicit multicast tree forwarding and (2) BIER forwarding in OpenFlow. They realize explicit multicast trees with OpenFlow group tables. To support BIER, they leverage MPLS headers to encode the BIER bit string, which limits the implementation to bit strings with a length of 20 bits, and therefore a maximum of 20 receivers. Rules with exact matches for bit strings are installed in the OpenFlow forwarding tables. When a packet with a BIER header arrives at a switch and a rule for its bit string is available, the packet can be immediately transmitted over the indicated interfaces. Otherwise, a local BIER agent running on the switch and maintaining the BIFT is queried. The local BIER agent installs a new flow entry for the specific bit string in the OpenFlow forwarding table. Thus, this approach requires bit string-specific state instead of IPMC group specific state. Furthermore, it is not likely to work well with quickly changing multicast groups as most subscription changes require configuration changes in the forwarding table of the switch. BIER with support for traffic engineering (TE) has been proposed in [39]. It uses the same header format but features different forwarding semantics and is not compatible with normal BIER. In [40] we have proposed and evaluated several FRR algorithms for BIER-TE and implemented them in a P4-based prototype [5].

5. Bit Index Explicit Replication (BIER)

First, we give an overview of BIER [2]. Afterwards, we present the Bit Index Forwarding Table (BIFT), which is the forwarding table for BIER. Then, we describe the BIER forwarding procedure.

5.1. Overview

We introduce essential nomenclature for BIER, the layered BIER architecture, the BIER header, and the BIER forwarding principle.

5.1.1. BIER Domain

BIER leverages a domain concept to transport IPMC traffic in a scalable manner, which is illustrated in Figure 2. Bit-Forwarding Routers (BFRs) forward BIER multicast traffic within the BIER domain. Inbound multicast traffic enters the domain through Bit-Forwarding Ingress Routers (BFIRs) and leaves it through Bit-Forwarding Egress Routers (BFERs). Border routers usually implement both BFIR and BFER functionality. When a BFIR receives an inbound IPMC packet, it pushes a BIER header onto the IPMC packet which indicates all BFERs that should receive a packet copy. BFRs utilize the information in the BIER header to forward BIER packets to all desired destinations along the paths induced by the interior gateway protocol (IGP). Thereby, packets are replicated if needed. Finally, the BFERs remove the BIER header before forwarding IPMC packets outside the domain.

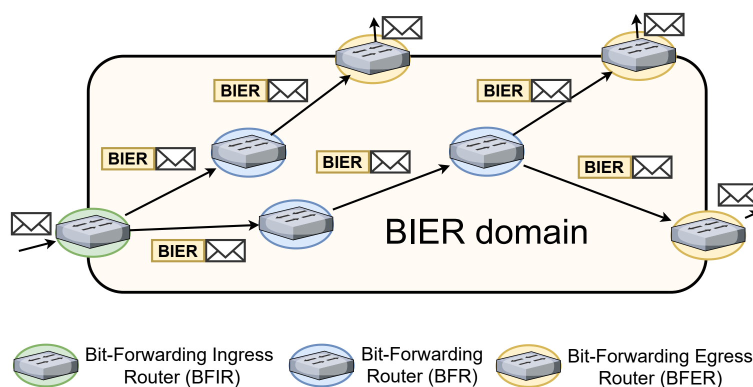


Figure 2: IPMC traffic enters a BIER domain through BFIRs which equip it with a BIER header. BFRs forwarded the traffic based on the BIER header within the domain on paths induced by the IGP. BFERs remove the BIER header when the traffic leaves the domain.

5.1.2. A Layered BIER Architecture

The BIER architecture can be subdivided into three layers: the IPMC layer, the BIER layer, and the routing underlay which is the forwarding mechanism for unicast traffic. In IP networks, the latter corresponds to the interior gateway protocol (IGP). Figure 3 shows the relation among the layers.

The IPMC layer requests multicast delivery for IPMC packets to a set of receivers in a BIER domain that depend on IPMC subscriptions. That is, when an inbound IMPC packet arrives at a BFIR, the BFIR equips the IPMC packet with an appropriate BIER header indicating all desired destinations. The BIER layer forwards these packets through the BIER domain to all receivers indicated

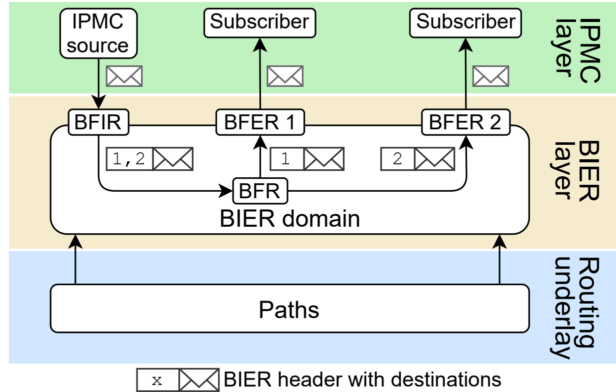


Figure 3: Layered BIER architecture with IPMC layer, BIER layer, and the routing underlay.

in the BIER header, thereby implementing a stateless point-to-multipoint tunnel für IPMC. The BIER layer leverages the forwarding information of the routing underlay to populate the forwarding tables of the BFRs. As a result, BIER traffic to a specific BFER follows the same path as unicast traffic towards that BFER. If two BFR are connected on Layer 2, the BIER traffic is directly forwarded; otherwise, the BFR neighbor is reachable only over the routing underlay so that the BIER traffic is encapsulated and forwarded over the routing underlay. When a BIER packet reaches a BFER that should receive a packet copy, the BFER removes the BIER header and passed the IPMP packet to the IPMC layer for further forwarding.

5.1.3. BIER Header and Forwarding Principle

The BIER header contains a bit string to identify BFERs. For brevity, we talk in the following about the BitString of a packet to refer to the bit string in the BIER header of that packet. The BitString is of a specific lenght. Each bit in the BitString corresponds to one specific BFER. The bits are assigned to BFERs starting with the least significant bit. BIER devices must support a BIER header of 256 bits. As this may not suffice to assign bits to all BFERs in large networks, the standard [2] defines subdomains to cope with that problem. This is a technical detail that we do not consider any further and our proposed solution can be easily adapted to subdomains.

When a BFIR receives an IPMC packet, it pushes a BIER header to the IPMC packet, determines the set of BFERs that must receive the traffic of the respective IMPC group, and activates in the BitString the bits corresponding to these BFERs. Packets are forwarded based on the information in their BIER header. A BFR sends a packet to any of its interfaces if at least one BFER indicated in the BIER header is reached in the routing underlay over this specific interface. To avoid duplicates, only those bits are kept in the BitString whose BFERs can be reached over the specific interface.

Figure 4 illustrates the BIER forwarding principle. It shows a small BIER domain with four nodes, each of them being BFR, BFIR, and BFER. Hosts are attached to all BIER nodes and participate in a single multicast group. Host 1 sends an IPMC packet to all other hosts. The figure visualizes how the BitString changes along the forwarding tree whose paths are inherited from the routing underlay.

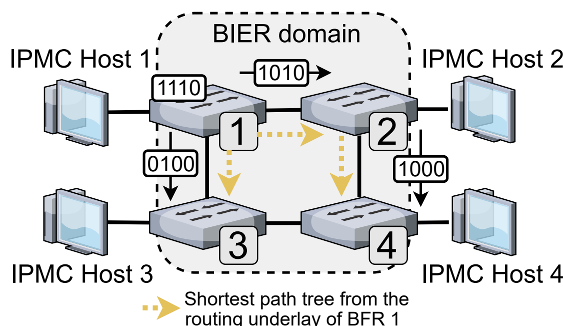


Figure 4: An IPMC packet is forwarded from Host 1 to all other hosts via a BIER domain. Within the domain, BIER packets are forwarded based on the BitString.

The information of the BIER forwarding tables depends only on the routing underlay. In Section 5.2 we explain the structure of the table and how its entries are calculated. In contrast to traditional IPMC forwarding, BIER forwarding does not require knowledge about IPMC groups. This has several advantages. BFRs do not neet to keep state per IPMC group. This frees core nodes of a BIER domain from signalling and state processing per IPMC group when subscriptions or routes change, e.g., in case of failures. This makes BIER forwarding in core nodes more robust and scalable than traditional IPMC forwarding. BFIRs still participate in IPMC signaling to keep track of group changes in order to adapt the BIER header for each IPMC group. BFERs forward outbound IPMC traffic in a traditional way.

5.2. Bit Index Forwarding Table

In this section we describe the Bit Index Forwarding Table (BIFT) which is the forwarding table of BFRs. We explain its structure and the computation of its entries.

First, we define BFR neighbors (BFR-NBRs) before we introduce the structure of the BIFT. The BFR-NBRs of a BFR A are those BFRs, that are adjacent to A according to the paths of the routing underlay.

Each BFR maintains a Bit Index Forwarding Table (BIFT) to determine the NH, i.e., BFR-NBR, when forwarding a packet. Table 1 shows the structure of the BIFT. For each BFER, the BIFT contains one entry which consists of a forwarding bitmask (F-BM) and the BFR-NBR to which the packet should be sent. The F-BM is used in the forwarding process to clear bits in a packet's BitString before transmission. The BFR-NBR for a BFER is derived as the

BFER	F-BM	BFR-NBR

Table 1: Header of the BIFT.

BFR-NBR on the path from the considered BFR to the BFER in the routing underlay. The F-BM for a BFER is composed as a bit string where all bits are activated that belong to BFERs with the same BFR-NBR. As a result, all BIFT entries with the same BFR-NBRs also have the same F-BM.

Table 2 illustrates the BIFT of BFR 1 in the example given in Figure 4.

BFER	F-BM	BFR-NBR
1	-	-
2	1010	2
3	0100	3
4	1010	2

Table 2: BIFT of BFR 1.

5.3. BIER Forwarding

In this paragraph we describe BIER forwarding. First, we explain the procedure how BIER processes packets. Then, we show a forwarding example. Finally, we illustrate the BIER header stack.

5.3.1. BIER Forwarding Procedure

BFRs process BIER packets in a loop. When a BFR receives a BIER packet, it determines the position of the least-significant activated bit in the BitString. The position of that bit corresponds to a BFER which is processed in this specific iteration of the loop. The BFR looks up that BFER in the BIFT, which results in a BFR-NBR and a F-BM. Then, a copy of the BIER packet is created for transmission to that BFR-NBR. Before transmission, all bits are cleared in the BitString of the packet copy that are not reachable through the same BFR-NBR. This is achieved by an AND-operation of the BitString and the F-BM. We denote this action as “applying the F-BM to the BitString”. Then, the packet copy is forwarded to the indicated BFR-NBR. All BFERs in the IPMC subtree of that BFR-NBR eventually receive a copy of this sent packet if their bit is activated in the BitString of the packet copy. Thus, all BFERs of this IMPC subtree can be considered as processed. Therefore, their bits are removed from the BitString of the remaining BIER packet. To that end, the BFR applies the complement of the F-BM to the BitString of the remaining BIER packet. This ensures that packets are delivered only once to intended receivers. If the BitString in the remaining BIER packet still contains activated bits, the loop restarts; otherwise the processing loop stops.

When the BFER that is currently processed corresponds to the BFR itself, the F-BM and BFR-NBR of its BIFT entry are empty. Then, a copy of the

BIER packet is created, the BIER header is removed, and the packet is passed to the IPMC layer within the BFR. Afterwards, the processed bit is cleared in the BitString of the remaining BIER packet, and the loop restarts if the BitString contains activated bits; otherwise the loop stops.

5.3.2. BIER Forwarding Example

We assume that BFIR 1 in Figure 4 receives an IPMC packet from IPMC Host 1 that should be sent to the IPMC Hosts 2, 3, and 4. Therefore, it adds a BIER header with the BitString `1110` to the IPMC packet and processes it. The least-significant activated bit corresponds to BFR 2. BFR 1 looks up the activated bit in its BIFT which is shown in Table 2. Then, it creates a packet copy and applies the F-BM to the BitString of that copy. This sets the BitString to `1010`. Then, the packet copy is forwarded to BFR 2. Aftwards, BFR 1 clears the activated bits of the F-BM from the BitString of the remaining original BIER packet. This leaves a packet with the BitString `0100`. BFR 1 processes the next activated bit, i.e. the bit for BFER 3. A packet copy is created, and the F-BM is applied which leaves the BitString `0100` in the packet copy. Then it is forwarded to BFR 3.

BFR 2 process the packet in the same way. As a result, it forwards one packet copy with the BitString `1000` to BFR 4. Additionally, it sends an IPMC packet without BIER header to its connected host. BFR 3 and 4 do the same when they receive their respective BIER packet.

5.3.3. BIER Header Stack

Without loss of generality, we assume in the following that the routing underlay is IP. Furthermore, we neglect the role of Layer 2 to facilitate readability.

Each BIER device is also an IP device. However, not every IP device is a BIER device. In Figure 5, the “Pure IP-node” is an IP node without BIER functionality. It belongs to the IP topology but not to the BIER topology. This influences the header stack of forwarded BIER packets. BFR 1, 2 and 3 are

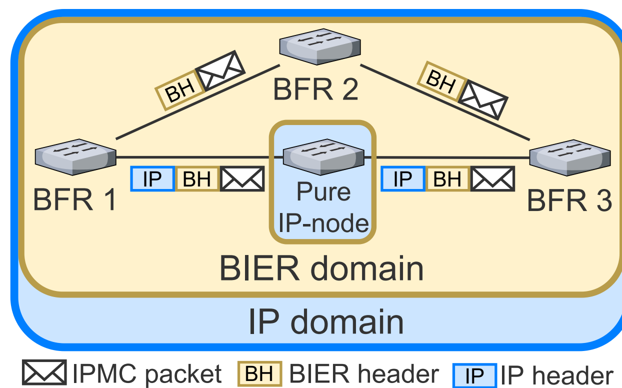


Figure 5: BIER traffic forwarded over pure IP nodes requires additional IP encapsulation.

both IP and BIER devices. The three BFRs are BFR-NBRs to each other. BFR 1 and 2 are neighbors to each other in both the IP and BIER topology because they are directly connected on Layer 2. Therefore, they exchange BIER packets on Layer 2 without an additional header. Since the pure IP node is not part of the BIER topology, BFR 1 and BFR 3 are BFR-NBRs although they are not neighbors in the IP topology. To exchange packets, BFR 1 and BFR 3 encapsulate BIER packets with an IP header and forward them via the pure IP node. When BFR 1 or 3 receive the packet, they remove the IP header and process the BIER header.

6. BIER Fast Reroute

The necessity for resilience mechanisms in BIER networks has been discussed in [41] without proposing any mechanism. In this section we introduce BIER fast reroute (BIER-FRR) to protect BIER traffic against link and node failures by taking advantage of reconvergence and FRR mechanisms of the routing underlay. We explain why regular BIER cannot protect BIER traffic sufficiently against failures, and present BIER-FRR for link and node protection, respectively. Finally, we discuss the protection properties of BIER-FRR.

6.1. Link Protection

In this paragraph we introduce BIER-FRR with link protection. First, we explain why relying on the available features of BIER and a resilient routing underlay is not sufficient for protection against link failures. Afterwards, we describe BIER-FRR with link protection and show a forwarding example.

6.1.1. Resilience Problems of BIER for Link Failures

BFR-NBRs may be directly connected over Layer 2 or they may be reachable only over Layer 3 so that IP encapsulation is needed for them to exchange BIER traffic (see Section 5.3.3). This has impact on the effect of link failures.

If neighboring BFRs are reachable only over Layer 3, they exchange BIER traffic IP-encapsulated towards each other. If a link on the path towards the BFR-NBR fails, the BFR-NBR is not reachable until the routing underlay has restored reachability. This may be due to IP-FRR, which is fast, or IP routing reconvergence, which is slow. In any case, the reachability on the BIER layer is also restored and no further action needs to be taken. Possibly, the path in the routing underlay changed, which may affect the neighboring relationships among BFRs, so that BIFTs need to be recomputed. This, however, is not time-critical.

If BFR-NBRs are directly connected over Layer 2, they exchange packets without an additional IP header. If the link between them is broken, protection mechanisms on Layer 3, in particular IP-FRR, cannot help because the BIER packet is not equipped with an IP header. Therefore, affected BIER traffic cannot be forwarded until a new BFR-NBR is provided in the BIFT for affected BFERs. Thus, the BIFT needs to be updated. This process takes time to

recompute the entries based on the new paths from the routing underlay and starts only after reconvergence of the routing underlay has completed. This is significantly later than FRR mechanisms on the routing underlay restore connectivity for unicast traffic.

BIER-FRR with link protection effects that a BFR affected by a link failure can forward BIER traffic again as soon as its connectivity problem is detected on the BIER layer and the routing underlay is able to forward unicast traffic again.

6.1.2. BIER-FRR with Link Protection

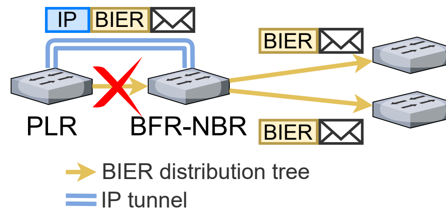


Figure 6: BIER-FRR with link protection is needed for BFR-NBRs which are directly connected on Layer 2: they IP-encapsulate BIER traffic towards a BFR-NBR after it is detected unreachable.

The concept of BIER-FRR with link protection is illustrated in Figure 6. BFRs must be able to detect link failures. This may happen, e.g., through loss of light detection or through bidirectional forwarding detection (BFD) with BFR-NBRs [42]. If an unreachable BFR-NBR is detected, a BFR IP-encapsulates BIER traffic towards that BFR-NBR. As a result, the BIER traffic will reach the affected BFR-NBR again as soon as reachability on the routing underlay is restored. This can be very fast if the routing underlay leverages FRR. When the traffic arrives at the BFR-NBR, the additional IP header is removed and packets are processed as normal BIER traffic.

6.1.3. Example for BIER-FRR with Link Protection

Figure 7 shows the BIER topology from the earlier forwarding example in Figure 4 with a link failure. For convenience, the BIFT of BFR 1 is displayed again in Table 3.

When BFR 1 sends a BIER packet to all other BFERs, the BitString is `1110`. First a packet copy is successfully deliverd to BFER 2 and BFER 4 so that the BitString of the remaining packet is `0100`, i.e., next a packet must be forwarded to BFER 3. However, BFR-NBR 3 is unreachable for BFR 1 due to the link failure. Therefore, BFR 1 IP-encapsulates the BIER packets towards BFR 3. As soon as the routing underlay restores connectivity, the IP-encapsulated BIER packets is detoured via BFR 2 and BFR 4 towards BFR 3. Thus, BIER-FRR with link protection may send a second packet copy over a link.

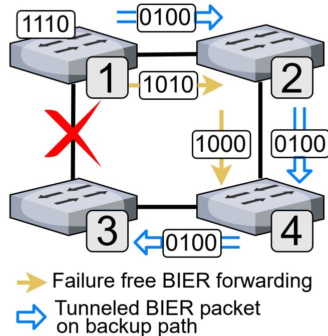


Figure 7: Packet paths and example topology for BIER-FRR with link protection.

BFER	F-BM	BFR-NBR
1	-	-
2	1010	2
3	0100	3
4	1010	2

Table 3: BIFT of BFR 1.

6.2. Node Protection

We introduce BIER-FRR with node protection. First, we discuss why regular BIER cannot protect BIER traffic sufficiently fast against node failures. Afterwards, we present the concept of BIER-FRR with node protection, extend the BIFT with backup entries, show a forwarding example, and explain how backup entries are computed.

6.2.1. Resilience Problems of BIER for Node Failures

If a BFR fails, all downstream BFRs are unreachable. This problem cannot be quickly repaired by the routing underlay because traffic directed to the failed node cannot be delivered. Thus, alternate BFR-NBRs are needed. These are provided when the routing underlay has reconverged and the BIFT entries are recomputed. This, however, may take long time.

BIER-FRR with node protection shortens this time so that affected BIER traffic can be delivered in the presence of node failures as soon as connectivity for unicast traffic is restored in the routing underlay.

6.2.2. BIER-FRR with Node Protection

We propose BIER-FRR with node protection to deliver BIER traffic even if the BFR-NBR fails. The concept is shown in Figure 8. When a PLR cannot reach a BFR-NBR, it tunnels copies of the BIER packet to all BFR next-next-hops (BFR-NNH) in the distribution tree that should receive or forward a copy. Thus, for each such BFR-NNH, an individual packet copy is created. The packet is then tunneled to the BFR-NNH with an additional header of the routing underlay; these packets are delivered as soon as the routing underlay restores connectivity. When the BFR-NNH receives such a packet, it removes the tunnel header and processes the resulting BIER packet.

If a BFR-NBR is unreachable, the link towards the BFR-NBR or the BFR-NBR itself may have failed. Therefore, the BFR-NBR should also receive a packet copy encapsulated by the routing underlay.

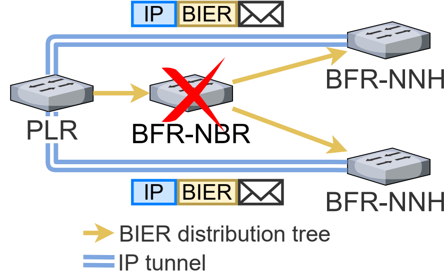


Figure 8: Concept of BIER-FRR with node protection.

BFER	F-BM	BFR-NBR
1	primary F-BM backup F-BM	primary NH backup NH
...	...	...

Table 4: Structure of a BIFT with backup entries.

When a packet copy is sent to multiple BFR-NNHs instead of the BFR-NBR, the the BitString of the forwarded packet copies must be modified appropriately to avoid duplicate packets at BFERs. These modifications can be obtained with backup F-BMs, which is explained in more detail in Section 6.2.5.

6.2.3. BIFT with Backup Entries

To support BIER-FRR with node protection, the BIFT must be extended with backup entries. The structure of a BIFT with backup entries is shown in Table 4.

The normal BIFT entries are called primary entries. The backup entries have the same structure as the primary entries. When a BFR-NBR is reachable, the primary entries are used for forwarding. If a BFR-NBR becomes unreachable, the corresponding backup entry is used for forwarding in the same way as a primary entry with only one difference. The packet is not forwarded natively but instead it is always tunneled to the backup NH through the routing underlay.

6.2.4. Example for BIER-FRR with Node Protection

Figure 9 shows an example topology and Figure 10 illustrates the distribution tree for BFR 1 and BFR 2 based on the paths from the routing underlay. Table 5 shows the BIFT of BFR 1 with primary and backup entries.

We illustrate the forwarding with BIER-FRR when BFR 2 fails. We assume that BFR 1 needs to send a BIER packet to BFR 6, i.e. the packet contains the BitString `100000`. As BFR 2 is unreachable, the primary NH of BFR 1 to BFR 6, which is BFR 2, cannot be used. Therefore, the backup entry is utilized. That means, the backup F-BM `101000` (see Table 5) is applied to the copy of the BIER packet and then it is tunneled through the routing underlay to the backup NH BFR 4. BFR 1 applies the complement of the backup F-BM

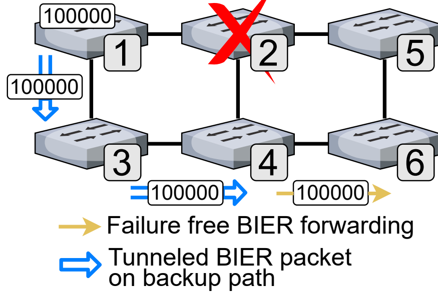


Figure 9: A packet is sent from BFR 1 to BFR 6 over a backup path using node protection.

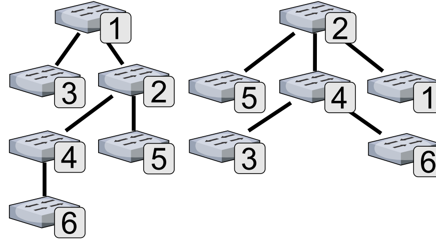


Figure 10: Shortest-path tree of BFR 1 and BFR 2.

BFER	F-BM	BFR-NBR
1	000001 -	- -
2	111010 000010	2 2
3	000100 000100	3 3
4	111010 101000	2 4
5	111010 010000	2 5
6	111010 101000	2 4

Table 5: BIFT of BFR 1 with primary and backup entries.

`010111` to the BitString of the original BIER packet which is then `000000`. As the BitString of the remaining BIER packet has no activated bits anymore, the forwarding process terminates at BFR 1.

The routing underlay delivers the packet copy from BFR 1 to BFR 4 as soon as connectivity is restored. BFR 4 removes the tunnel header and forwards the BIER packet to BFR 6.

If the BitString of the packet was `100100`, i.e., BFER 3 should have received a copy of the packet, too, a regular BIER packet would have been forwarded directly to BFR 3 before BIER-FRR tunnels another copy of the BIER packet to BFR 4. Thus, BIER-FRR with node protection may increase the traffic on a link to ensure that all relevant NNHs receive a packet copy.

6.2.5. Computation of Backup Entries

We compute backup NHs and backup F-BMs for BFERs at a specific BFR which we call PLR in this context. To that end, we distinguish two cases: the

BFER is not a BFR-NBR (1) or it is a BFR-NBR (2).

In the first case, the BFER is reached from the PLR through the routing underlay via a considered NH and next-next-hop (NNH). The considered NNH becomes the backup NH for the BFER. The corresponding backup F-BM requires activated bits for a set of BFERs. This set comprises all BFERs whose paths in the routing underlay from the PLR also traverses the considered NH and NNH. This F-BM can be computed by bitwise AND'ing the PLR's F-BM for the considered BFER and the considered NH's F-BM.

In the second case, the considered BFER is a BFR-NBR. Then, the NH is also taken as backup NH. This ensures that the NH receives a copy of the BIER packet if the NH cannot be reached due to a link failure. To avoid that the NH distributes further packet copies, the backup F-BM contains only the activated bit for the considered BFER.

We illustrate both computation rules by an example. We consider the BIFT of BFR 1 in Table 5. The backup entry of BFER 6 is an example for the first computation rule. The backup NH for BFR 6 is BFR 4 as it is the NNH of BFR 1 on the path towards BFR 6 in Figure 10. The BFERs reachable from the PLR through BFR 4 are BFER 4 and BFER 6. Therefore, the backup F-BM is `101000`. It can be obtained by bitwise AND'ing the F-BM of BFR 1 for BFER 6 (`111010`) and the F-BM of BFR 2 for BFER 6 (`101100`). The latter can be derived from the multicast subtree of BFR 2 in Figure 10.

The backup entry of BFER 2 is an example for the second computation rule. The backup NH for BFER 2 is BFR 2 and the F-BM contains only one activated bit for BFER 2 (`000010`).

6.3. Properties of BIER-FRR

We have argued that restoration of BIER connectivity may take long time in case of a link failure since this process can start only after the reconvergence of the routing underlay has completed. To shorten the outage time, we introduced BIER-FRR which restores connectivity on the BIER layer as soon as unreachable BFR-NBRs are detected and the connectivity in the routing underlay is restored.

The general concept of BIER-FRR is simple: it requires some sort of detection that a BFR-NBR is no longer reachable, but it does not require any additional signalling as it is a local mechanism. Furthermore, it leverages the restoration of routing underlay so that BIER traffic can profit from FRR mechanisms in the routing underlay. It does not define alternate paths on the BIER layer, which is in contrast to another solution reported in [43].

BIER-FRR comes in two variants: link protection and node protection. Link protection is simple, it just encapsulates BIER traffic into a header of the routing underlay, but it cannot protect against node failures. The encapsulated packet may be sent over an interface over which also a regular copy of the same BIER-packet is transmitted. That means, up to two packet copies can be transmitted over at most one link in case of a failure, which runs in contrast to the actual idea of multicast.

Node protection is more complex. It requires a PLR to send backup copies of a BIER packet to all relevant NNHs encapsulated with a header of the routing underlay. This requires extensions to the BIFT for backup entries. However, it protects against link and node failures. The encapsulated packets may be sent over interfaces over which also a regular copy of the same BIER packet is transmitted. That means that even multiple packet copies can be transmitted over several links in case of a failure.

BIER-FRR is designed for single link and node failures. In case of multiple failures, BIER-FRR suffers from potential shortcomings of the routing underlay to cope with multiple failures, too, so that some traffic may be lost until the BIFT is updated. Furthermore, if both a NH and a NNH fail, the subtree of the NNH is no longer reachable until the BIFTs are updated. Some FRR techniques may cause routing loops in case of multiple failures [12]. In contrast, BIER-FRR cannot cause routing loops because it just leverages the routing underlay and does not propose new paths in failure cases.

6.4. Application of IP-FRR Mechanism on BIER Layer

In Section 3.1 we introduced IP-FRR and described LFAs. In [43] we discussed the application of LFAs on the BIER layer, i.e., in addition to the primary BFR-NBR, the BIFT contains a backup BFR-NBRs respectively, to which a BIER packet is forwarded when the primary NH is unreachable. We identified two major disadvantages. First, their application leaves a significant amount of BFERs unprotected against link or node failures because LFAs cannot guarantee full protection coverage [12]. This holds in particular when node protection is desired for which protection coverage is even lower than for link protection. Second, LFAs on the BIER layer introduce new paths in the BIER topology, which can cause rerouting loops for BIER traffic. Third, this approach assumes IP with IP-FRR as routing underlay while our approach works with any routing underlay and FRR mechanism. Therefore, we argue that the application of IP-FRR mechanisms on BIER layer is not sufficient for appropriate protection.

7. Introduction to P4

This section serves as a primer for readers who are not familiar with P4. First, we explain the general P4 processing pipeline. Then, we describe the concept of match+action tables, control blocks, and metadata. Finally, we explain the recirculate and clone operations.

7.1. P4 Pipeline

P4 is a high-level language for programming protocol-independent packet processors [44]. Its objective is a flexible description of data planes. It introduces the forwarding pipeline shown in Figure 11. A programmable parser reads packets and stores their header information in header fields which are carried together with the packet through the pipeline. The overall processing model is composed of two stages: the ingress and the egress pipeline with a packet buffer

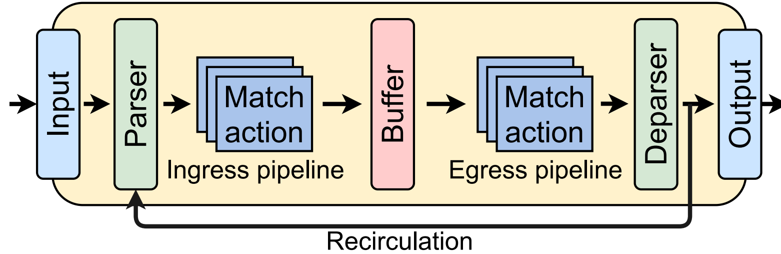


Figure 11: P4 abstract forwarding pipeline according to [44].

in between. The egress port of a packet has to be specified in the ingress pipeline. If no egress port has been specified for a packet at the end of the egress pipeline, the packet is dropped. At the end of the egress pipeline, a deparser constructs the packet with new headers according to the possibly modified header fields. P4 supports the definition and processing of arbitrary headers. Therefore, it is not bound to existing protocols.

7.2. Metadata

Metadata constitute packet-related information. There are standard and user-defined metadata. Examples for standard metadata are ingress port or reception time which are set by the device. User-defined metadata store arbitrary data, e.g., processing flags or calculated values. Each packet carries its own instances of standard and user-define metadata through the P4 processing pipeline.

7.3. Match+Action Tables

Match+action tables are used within the ingress and egress pipeline to apply actions to specified packets. The P4 program describes the structure of each match+action table. The rules are the contents of the table and are added to the table during runtime.

As match+action tables are essential for the description of our prototype, we introduce a compact notation for them by an example. The example is given in Figure 12. The table has the name "MAT_Simple_IP" and describes an implementation of simplified IP forwarding with match+action tables. In the following we use the prefix "MAT_" for naming MATs.

7.3.1. Match Part

A table defines a list of match keys that describe which header fields or metadata are used for matching a packet against the table. The match type indicates the matching method. P4 supports several match types: exact, longest-prefix (lpm), and ternary. The latter features a wildcard match. In our example in Figure 12, the match key is the destination IP address and lpm matching is applied.

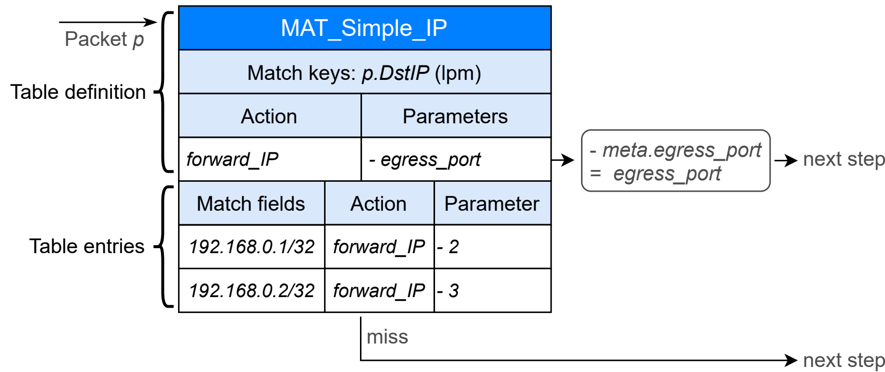


Figure 12: Match+action table for simplified IP forwarding.

7.3.2. Actions

The table further defines a list of actions including their signature which can be used by rules in case of a match. Actions are similar to functions in common programming languages and consist of several primitive operations. Inside an action further actions can be executed. Actions can modify header fields and metadata of a packet. In our example, this is the *forward_IP* action that requires the appropriate egress port as a parameter. Each action is illustrated by a flow chart on the right side of the table.

7.3.3. Rules

During runtime, the match+action tables can be filled with rules through an application programming interface (API). The rules contain match fields which are patterns that are to be matched against a packet's context selected by the match keys. In our example, the match fields are IP addresses. The rules further specify an action in the table definition and suitable parameters which are applied to the packet in case of a match.

In our example in Figure 12 we install two rules. In the first one, the match field is the IP address *192.168.0.1* and it applies the action *forward_IP* with the parameter *2*. This will send packets with the destination IP *192.168.0.1* over port *2*. The match field for the second rule is *192.168.0.2* and it sends the packet over port *3*. For all other destination IPs a miss occurs and no egress port is specified.

When describing match+action tables of our implementation in Section 8, we omit the actual rules as they are configuration data and not part of the P4 implementation.

7.4. Control Blocks

A control block consists of a sequence of match+action tables, operations and if-statements. They encapsulate functionality. Within control blocks other control blocks can be called. Both the ingress and egress pipeline are control

blocks that apply other control blocks. We use the prefix "CB_" for naming of our other control blocks. Examples of control blocks in our implementation are *CB_IPv4*, *CB_BIER*, or *CB_Ethernet*.

7.5. Recirculation

P4 does not support native loops. However, as indicated in Figure 11, the recirculation operation returns a packet to the beginning of the ingress pipeline. It activates a standard metadata field, i.e., a flag, which marks the packet for recirculation. The packet still traverses the entire pipeline and only at the end of the egress pipeline the packet is returned to the start of the ingress pipeline. When setting the *recirculate* flag, it is possible to specify which metadata fields should be kept during recirculation. All others are reset to their default values. In contrast, header fields modified during the processing remain modfied after recirculation. Another standard metadata field stores whether a packet has been recirculated.

7.6. Packet Cloning

P4 supports the packet cloning operation clone-ingress-to-egress (*CI2E*). *CI2E* can be called anywhere in the ingress pipeline. This activates the *CI2E* metadata flag which indicates that the packet should be cloned. However, the copy is created only at the end of the ingress pipeline. In the packet clone all header changes are discarded that have been made within the ingress pipeline. If *CI2E* has been called within the ingress pipeline, two packets enter the egress pipeline. One is the original packet that has been processed by the ingress control flow. The second packet is the copy without modifications from the ingress pipeline. Figure 13 illustrates this by an example.

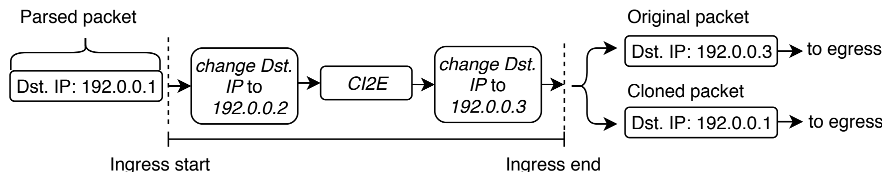


Figure 13: Illustration of the clone-ingress-to-egress (*CI2E*) operation: the destination IP of the clone is the one of the received packet although IP was modified before *CI2E* was called.

When the *CI2E* flag is set, it is possible to specify for the clone whether metadata fields should persist or be reset. When a packet clone enters the egress pipeline, an additional standard metadata flag identifies the packet as a clone. This allows different processing for original and cloned packets.

8. P4-Based Implementation of BIER and BIER-FRR

In this section, we describe the P4-based implementation of IP, IP-FRR, BIER, and BIER-FRR. We first describe the data plane followed by the control plane. In the end, we briefly explain our codebase.

8.1. Data Plane

First, we specify the handling of packet headers, then, we give a high-level overview of the processing pipeline, followed by a detailed explanation of applied control blocks.

8.1.1. Packet Header Processing

P4 requires that potential headers of a packet are defined a priori. Our implementation supports the header suite Ethernet/outer-IP/BIER/inner-IP. We use the inner IP header for regular forwarding and the outer IP header for FRR. During packet processing, headers may be activated or deactivated. Deactivated headers are not added by the deparser. *Encaps* actions in our implementation activate a specific header and set header fields. *Decaps* actions deactivate specific headers.

8.1.2. Overview of Ingress and Egress Control Flow

Figure 14 shows an overview of the entire data plane implementation which is able to perform IP and BIER forwarding as well as IP-FRR and BIER-FRR. It

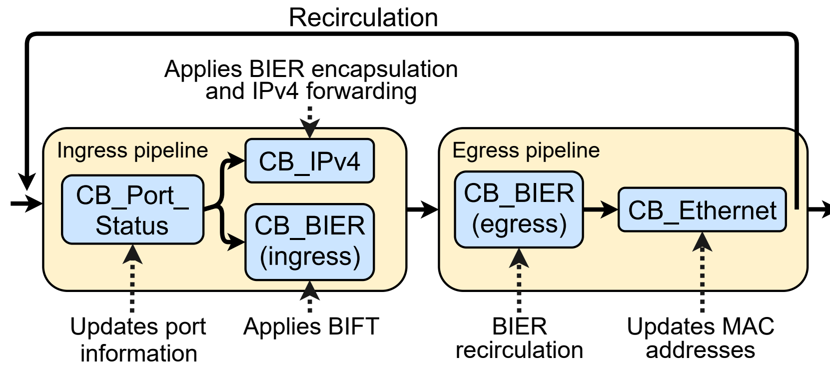


Figure 14: Overview of ingress and egress control flow.

is divided into ingress and egress control flow which are given as control blocks. In the ingress and egress control block the CB_IPv4 and CB_BIER control block are only applied to their respective packets, i.e., the CB_IPv4 control block is applied to IP packets and the CB_BIER control block is applied to BIER packets. We first summarize their operations and describe their implementation in detail in the following Sections.

When a packet enters the ingress pipeline, it is processed by the *CB_Port_Status* control block. It updates the port status (up/down) and records it in the user-defined metadata *meta.live_ports* of the packet. This possibly triggers FRR actions later in the pipeline. Then, the *CB_IPv4* control block or the *CB_BIER* control block is executed depending on the packet type.

The *CB_IPv4* control block is applied to both unicast and multicast IP packets. Unicast packets are processed by setting an appropriate egress port,

possibly using IP-FRR in case of a failure. IPMC packets entering the BIER domain are equipped with a BIER header and recirculated for BIER forwarding. IPMC packets leaving the BIER domain are forwarded using native multicast.

The *CB_BIER* control block is applied to BIER packets. There is a *CB_BIER* control block for the ingress control flow and another for the egress control flow. A processing loop for BIER packets is implemented which extends over both *CB_BIER* control blocks. At the beginning of the processing loop in the ingress flow the BitString is copied to metadata *meta.remaining_bits*. This metadata is used to track for which BFERs a copy of the BIER packet still needs to be sent. Then, rules from the *MAT_BIFT* are applied to the packet. This also comprises BIER-FRR actions which encapsulate BIER packets with an IP header if necessary. Within these procedures, the BIER packet is cloned so that the original packet and a clone enter the egress control flow. The processing loop stops if the *meta.remaining_bits* are all zero.

In the *CB_BIER* control block of the egress control flow, the *recirculate* flag is set for cloned packets. At the end of the egress control flow, the clone is recirculated to the ingress control flow with modified *meta.remaining_bits* to continue the processing loop. The non-cloned BIER packet is just passed to the *CB_Ethernet* control block.

The *CB_Ethernet* control block updates the Ethernet header of each packet. Then, the packet is sent if an egress port is set and the *recirculate* flag has not been activated. If the *recirculate* flag is activated, the packet is recirculated instead. This applies to cloned BIER packets in the processing loop or to packets that require a second pass through the pipeline: BIER-encapsulated IPMC packets, BIER-decapsulated IPMC packets, IP-encapsulated BIER packets, or IP-decapsulated BIER packets. If neither *recirculate* flag is activated and nor the egress port is set, the packet is dropped.

8.1.3. CB_Port_Status Control Block

The control block *CB_Port_Status* records whether a port is up or down in the user-defined metadata *meta.live_ports* of a packet. Figure 15 shows that it consists of only the match+action table *MAT_Port_Status*.

The table does not define any match keys. As a result, the first entry matches every packet. We install only a single rule which calls the action *set_port_status*. It copies the parameter *live_ports* to the user-defined metadata *meta.live_ports*. *Meta.live_ports* is a bit string where each bit corresponds to a port of the switch. If the port is currently up, the bit is activated, otherwise, the bit is deactivated. The metadata field *meta.live_ports* is later used by both the *CB_IPv4* and *CB_BIER* control block to decide whether IP-FRR and BIER-FRR should be applied. The parameter *live_ports* in the table is updated by the local controller when the port status changes, which will be explained in Section 8.2.1.

8.1.4. CB_IPv4 Control Block

The *CB_IPv4* control block handles IPv4 packets. Its operation is shown in Figure 16.

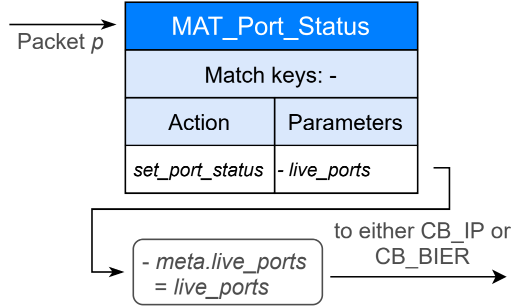


Figure 15: In the control block *CB_Port_Status* the table *MAT_Port_Status* copies the information about live ports to the user-defined metadata field *meta.live_ports* of the packet.

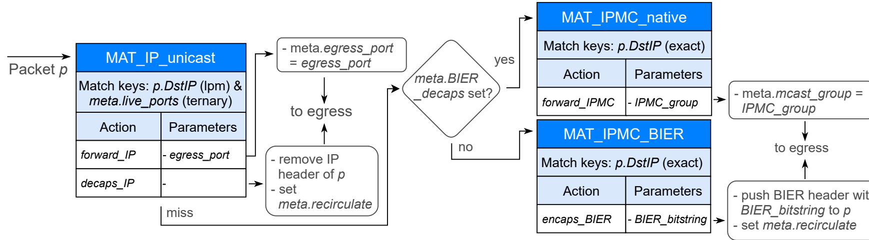


Figure 16: The *CB_IPv4* control block handles IPv4 packets.

It leverages three match+action tables: *MAT_IP_unicast*, *MAT_IPMC_native*, and *MAT_IPMC_BIER*. Packets are processed by these tables depending on their type. *MAT_IP_unicast* performs IP unicast forwarding including IP-FRR. IPMC packets encounter a miss and are relayed by the control flow to *MAT_IPMC_native* or *MAT_IPMC_BIER*. *MAT_IPMC_native* performs native multicast forwarding for IPMC packets leaving the BIER domain while *MAT_IPMC_BIER* just adds a BIER header for IPMC packets entering the BIER domain.

MAT_IP_unicast. This match+action table uses the IP destination address and the metadata *meta.live_ports* as match keys. The IP destination address is associated with a longest prefix match and the *meta.live_ports* with a ternary match. We first explain our implementation of IP-FRR. The rules contain an IP prefix and a *required_port* pattern as match fields (not shown in the table). *Required_port* corresponds to a bit string of all egress ports and is a wildcard expression with only a single zero or one for the primary egress port of the traffic, i.e., `*...*0*...*` or `*...*1*...*`. If FRR is desired for an IP prefix, two rules are provided: a primary rule with `*...*1*...*` as *required_port* pattern, and a backup rule with `*...*0*...*`.

The table offers two actions: *forward_IP* and *decaps_IP*. We explain both in the following in detail.

The *decaps_IP* action is applied to packets that are addressed to the node

itself. For such rules the *required_port* pattern is set to *...*. Those IP packets are typically BIER packets that have been encapsulated in IP by other nodes for BIER-FRR. Therefore, the IP header is removed and the *recirculate* flag is set so that the packet can be forwarded as BIER packet in a second pass of the pipeline. In theory, other IP packets with the destination IP addresses of the node itself may have reached their final destination. They need to be handed over to a higher layer within the node. However, this feature is not required in our prototype so that we omit it in our implementation.

The *forward_IP* action is applied for other unicast address prefixes and requires an *egress_port* as parameter. It sets the *meta.egress_port* to the indicated egress port so that the packet is switch-internally relayed to the right egress port. The IP-FRR mechanism as explained above may be used in conjunction with *forward_IP* to provide an alternate egress port when the primary egress port is down. This mechanism allows implementation of LFAs.

IPMC addresses encounter a miss in this table so that their packets are further treated by the control flow in the *CB_IPv4* control block. It checks whether the *meta.BIER_decaps* bit has been set. If so, the IPMC packet came from the BIER domain and has been decapsulated. Therefore, it is relayed to the *MAT_IPMC_native* table for outbound IPMC traffic. Otherwise, the IMPC packet has been received from a host and requires forwarding through the BIER domain. Therefore, it is relayed to the *MAT_IPMC_BIER* table.

MAT_IPMC_native. This match+action table implements native IPMC forwarding. It is used by a BFER to send IPMC packets to hosts outside the BIER domain that have subscribed to a specific IPMC group. The table *MAT_IPMC_native* uses the IP destination address as match key with an exact match. It defines only the *forward_IPMC* action and requires a switch-internal multicast group as parameter, which is specific to the IPMC group (IP destination address) of the packet. The action sets this parameter in the *meta.mcast_group* of the packet. As a consequence, the packet is processed by the native multicast feature of the switch. This results in packet copies for every egress port contained in the switch-internal multicast group *meta.mcast_group* with the corresponding egress port set in the metadata of the packets. The set of egress ports belonging to that group can be defined through a target-specific interface, which is done by the controller in response to received IGMP packets. Packets encountering a miss in this table are dropped at the end of the pipeline.

MAT_IPMC_BIER. This match+action table uses the IP destination address as match key with an exact match. It defines only the *encaps_BIER* action and requires the bit string as parameter, which is specific to the IPMC group (IP destination address) of the packet. The action pushes a BIER header onto the packet and sets the specified BitString. Then the *recirculate* flag is set so that the packet can be forwarded as a BIER packet in a second pass of the pipeline. Packets encountering a miss in this table are dropped at the end of the pipeline.

8.1.5. CB_BIER Control Block

The *CB_BIER* control block processes BIER packets. It is illustrated in Figure 17.

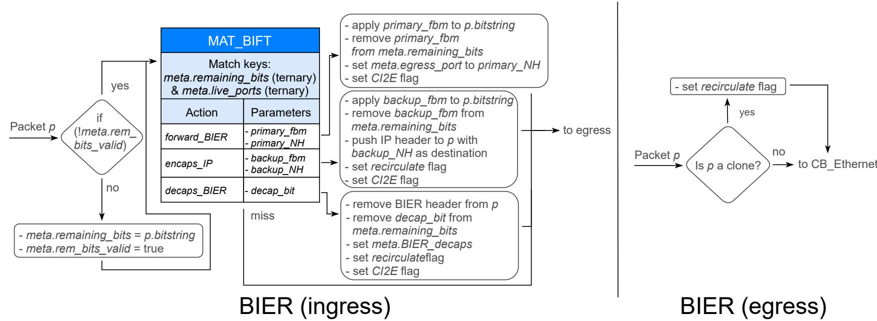


Figure 17: The *CB_BIER* control blocks in the ingress and egress pipeline implement BIER fowarding as a processing loop.

The user-defined metadata *meta.remaining_bits* is used during BIER processing to account for the BFERs that still need a copy of the packet. It serves as a control variable for the processing loop. When a BIER packet is processed by the *CB_BIER* control block for the first time, *meta.remaining_bits* is initialized with the BitString of the packet. The user-defined metadata *meta.remaining_bits_valid* is initially zero. It is activated after *meta.remaining_bits* is initialized and prevents overwriting
meta.remaining_bits when the packet is recirculated.

Then the match+action table *MAT_BIFT* is applied. It implements BIER forwarding including BIER-FRR according to the principle we developed for IP-FRR in Section 8.1.4. Match keys are the packet's *meta.remaining_bits* indicating BFERs, and *meta.live_ports* indicating live egress ports. The match types are ternary. Rules are provided for all individual BFERs both for failure-free cases and failure cases. The match field of these rules consists of two bit strings that we call *dest_BFER* and *required_port* (not shown in the table). The *dest_BFER* bit string has the bit position for the respective BFER activated and all other bit positions set to wildcards (`*...*1*...*`). The *required_port* bit string is used as in Section 8.1.4 to select between primary and backup rules. In case of a match, there are three possible actions.

Decaps_BIER is called by the rule whose activated bit in *dest_BFER* refers to the node itself. It has a F-BM with only the bit of the BFER activated and no primary or backup NH. If this rule matches, the node should receive a copy of the packet. The action removes the BIER header of the packet, activates the user-defined metadata flag *meta.BIER_decaps*, and the *recirculate* flag so that the resulting IPMC packet is processed in a second pass of the pipeline. In addition, the complement of F-BM is used to clear the bit for the processing node itself in *meta.remaining_bits*.

Forward_BIER is called by rules whose activated bit in *dest_BFER* refers to

other nodes and where the *required_port* bit string indicates that the egress port works. Thus, *forward_BIER* is used for primary forwarding. It has the primary F-BM and the primary NH (egress port) as parameters. The primary F-BM is applied to clear bits from the BitString of the packet and the complement of the backup F-BM is applied to *meta.remaining_bits*. In addition, *meta.egress_port* is set to the primary NH.

Encaps_IP is called by rules where the *required_port* bit string indicates that the primary egress port does not work for the BFER specified in *dest_BFER*. Thus, *encaps_IP* is used for backup forwarding. It has the backup F-BM and the backup NH (IP address) as parameters. The backup F-BM is applied to clear bits from the BitString of the packet and the complement of the backup F-BM is applied to *meta.remaining_bits*. Then, an IP header is pushed with the destination address of the backup NH. The *recirculate* flag for the packet is activated as it requires IP forwarding in a second run through the pipeline.

At the end of *decaps_BIER*, *forward_BIER*, and *encaps_IP*, a flag for *CI2E* is set. This effects that a packet copy is generated at the end of the ingress pipeline. For the copy (clone), the *recirculate* flag is activated in the *CB_BIER* control block in the egress control flow. With this packet, the BIER processing loop continues. The *meta.remaining_bits* information must be kept to account for the BFERs that still need a packet copy.

When packets enter the *MAT_BIFT* table with *meta.remaining_bits* equal to zero, they encounter a miss. As a result, they are dropped at the end of the pipeline, which stops the processing loop for these BIER packets.

8.1.6. CB_Ethernet Control Block

The *CB_Ethernet* control block is visualized in Figure 18.

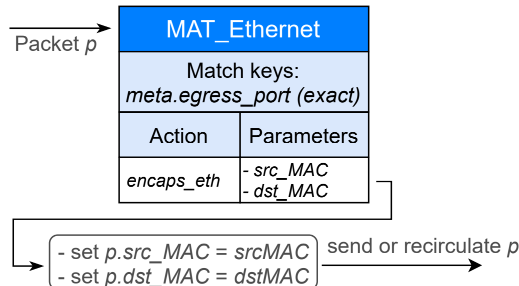


Figure 18: *CB_Ethernet* control block.

It applies the match+action table *MAT_Ethernet* to all packets. The match key is the egress port of the packet and the match type is exact. Only the action *encaps_eth* is defined which requires the parameters *src_MAC* and *dst_MAC*. It updates the Ethernet header of the packet by setting the source and destination MAC address which are provided as parameters. Rules are added for every egress port.

This behavior is sufficient as we assume that any hop is an IP node. Although MAC addresses are not utilized for packet switching, they are still necessary as packet receivers in Mininet discard packets if their destination MAC address does not match their own address.

8.2. Control Plane Architecture

The control plane is visualized in Figure 19. It consists of one global con-

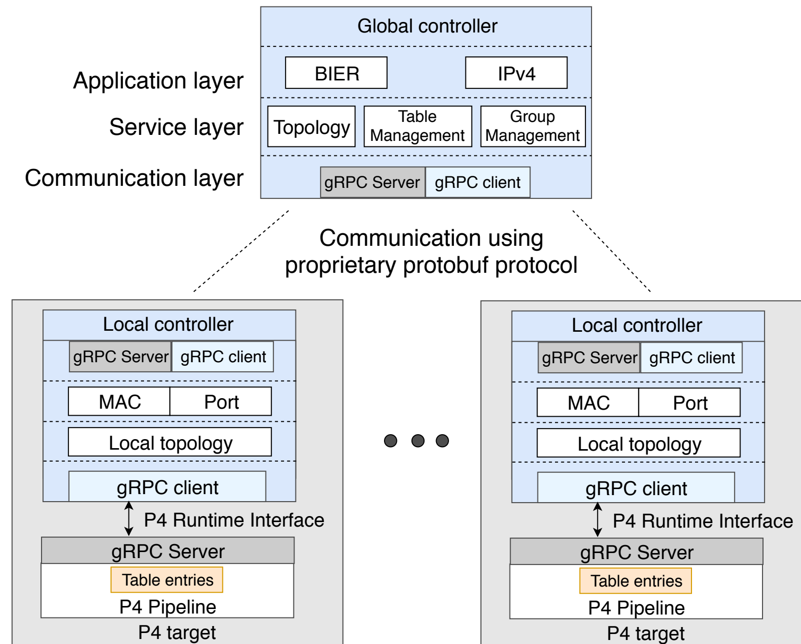


Figure 19: Controller architecture.

troller and one local controller per switch. The local controllers run directly on the switch hardware as P4 switches are mostly whiteboxes. The local controller takes care of tasks that can be performed locally while the global controller is in charge of configuration issues that require a global network view. In theory, a single controller could perform all tasks. However, there are three reasons that call for a local controller: scalability, speed, and robustness. Performing local tasks at the local controller relieves the global controller from unnecessary work. A local controller can reach the switch faster than a global controller. And, most important, a local controller does not need to communicate with the switch via a network. In case of a network failure, the local controller still reaches the switch while the global controller may be unable to do so. Local controllers have also been applied for similar reasons in LoCoSDN [45], P4-MACSec [46], and P4-IPSec [47]. In the following we explain the local and global controller in more detail.

8.2.1. Local Controller

Each switch has a local controller. Switch and local controller communicate via the so-called P4 Runtime which is essentially the Southbound interface in the SDN context. The P4 Runtime uses a gRPC channel and a protobuf-based communication protocol. It allows the controller to write table entries on the switch.

Figure 19 shows that the local controller keeps information about the local topology, learns about neighboring nodes, and port status, and configures this information in the tables of the switch. Moreover, it relays some packets to the global controller and writes table entries as a proxy for the global controller.

We leverage the local controller for three local tasks that we describe in the following: IGMP handling, neighbor discovery, and port monitoring.

IGMP Handling. Multiple hosts are connected to a switch. They leverage the Internet Group Message Protocol (IGMP) to join and leave IPMC groups. If the switch receives an IGMP packet, it forwards it to its local controller which then configures the switch for appropriate actions. For example, it adds a new host to the IPMC group and configures the native IPMC feature of the switch to deliver IPMC packets to the hosts. That feature is used only for carrying multicast traffic from the switch to the hosts. To populate the *MAT_IPMC_native* table, the local controller utilizes the Thrift channel instead of the P4 Runtime as this API is target-specific.

Neighbor Discovery. For neighbor discovery, we implemented a simple proprietary topology recognition protocol. All nodes announce themselves to their neighbors. It allows the local controller to learn the MAC address of the neighbor for each egress port. The local controller stores this information in the match+action table *MAT_Ethernet* which is utilized in the *CB_Ethernet* control block (see Section 8.1.6).

Port Monitoring. A P4 switch by itself is not able to find out whether a neighboring node is reachable. However, a fast indication of this information is crucial to support FRR. In a real network a local controller may test for neighbor reachability, e.g., using a BFD towards all neighbors, loss-of-light, loss-of-carrier, or any other suitable mechanism. Then, the local controller configures this information as a bit string in the match+action table *MAT_Port_Status* of the switch whenever the port status changes. Failure detection is target-dependent and out of scope of this document. Therefore we trigger failure processing of the local controller manually with a software signal. The local controller then activates IP-FRR and BIER-FRR if enabled and notifies the global controller for recomputation of forwarding entries.

8.2.2. Global Controller

We divide the architecture of the global controller in three layers: communication, service, and application (see Figure 19).

The communication layer is responsible for the communication with the local controllers. Each switch is connected to its local controller. Since the P4 runtime only allows one controller with write access, the global controller cannot directly control the switches. Therefore, it communicates with the local controllers to configure the switches. All changes calculated by the global controller are sent to the local controller using a separate channel. The local controller forwards the changes to the switch using the P4 runtime interface.

The service layer provides services for the application layer. This includes information about the topology, multicast groups, and entries in the tables on the switches. The application layer utilizes that information to calculate the table entries.

The global controller receives IGMP messages and keeps track of subscriptions to IPMC groups. If a host is the first to enter or the last to leave an IPMC group at a BFER, the global controller configures the *MAT_IPMC_BIER* table of all BFIRs with an appropriate bit string for the specific IPMC group by activating or deactivating the corresponding bit of the BFER. As a result, the BFIR starts or stops sending traffic from this IPMC group to the BFER.

The global controller sets all entries in the *MAT_IP_unicast* and *MAT_IPMC_BIER* tables of all switches and the entries in the *MAT_BIFT*s. If the global controller is informed by a local controller about a failure, it first reconfigures the *MAT_IP_unicast* and *MAT_IPMC_BIER* tables and then the entries of the *MAT_BIFT*s accordingly.

8.3. Codebase

The implementation of the BIER data plane and control plane including a demo can be downloaded at https://github.com/uni-tue-kn/p4-bier. The provided code contains a more detailed documentation of the BIER(-FRR) implementation. The demo contains several Mininet network topologies that were used to verify the functionality of BIER(-FRR). One of them is described in Section 9.1. Links can be disabled using Mininet, which enables the verification of the BIER-FRR mechanism. A simple host CLI allows multicast packets to be sent and incoming multicast packets to be displayed.

9. Evaluation

In this section we illustrate that BIER traffic is better protected with BIER-FRR. To that end, we conduct experiments in a testbed using our prototype. We first explain the experimental setup, the timing behavior of our emulation and our metrics. Finally, we describe the testbed setup and present experimental protection results in an BIER/IP network with and without IP-FRR and BIER-FRR, for link protection and node protection, respectively.

9.1. Methodology

First, we describe the general approach for our evaluation. Then, we discuss the timing behavior of a software-based evaluation. As the prototype switch is

differently controlled than typical routers, we adapt reaction times of the controller after a failure to mimic the timely behaviour of updates for IP forwarding tables and BIFTs. Finally, we explain our metrics.

9.1.1. General Setup

We emulate different topologies in Mininet [48]. The core network is implemented with our P4-based prototype and the software-based *simple_switch* which is based on the BMv2 framework [49]. It forwards IP unicast, IP multicast, and BIER traffic. One source and several subscribers are connected to the core network. The source periodically sends IP unicast and IP multicast packets. IP unicast packets are forwarded as usual through the core network. When IP multicast packets enter the core network, they are encapsulated with a BIER header at the BFIR. BFERs remove BIER headers and forward the IP multicast packets to the subscribers.

Rules for the match+action tables are computed by the global controller in an initial setup phase. In different scenarios we simulate link and node failures and observe packet arrivals at the subscribers. We study different combinations of IP-FRR and BIER-FRR to evaluate the delay until subscribers receive traffic again after a failure has been detected. Also in those cases, the local controller notifies the global controller to perform IP reconvergence and BIFT recomputation because FRR is meant to be only a temporary measure until the global forwarding information base has been updated as a response to the link or node failure.

We report events at the PLR and at all subscribers before and after the failure. For the PLR we show the following signals: failure detection at t_0, updates of IP forwarding entries, and updates of BIFT entries. For the subscribers we record receptions of unicast and multicast packets.

9.1.2. Timing Behavior

Our switch implementation in a small, virtual environment has a different timing behavior than a typical router in a large, physical environment. In particular signaling can be executed with insignificant delay in our virtual environment, e.g., notifying the global controller about the failure or the distribution of updated forwarding entries. This is different with routers and routing protocols in the physical world. Signaling requires significant time as routing protocols need to exchange information about the changed topology. Routers compute alternative routes and push them to their forwarding tables. Only after all unicast paths have been recomputed and globally updated by the routing underlay, BFRs can compute new forwarding entries for BIER and push them to their BIFTs. Thus, the BIFT is updated only significantly later compared to the unicast forwarding information base. To respect that in our evaluation, we configure the global controller to install new IP forwarding entries on the switches only after 150 ms after being informed about a failure and new BIFT entries another 150 ms later.

9.1.3. Metric

We perform experiments with and without IP-FRR and BIER-FRR, and compare the time after which unicast and multicast traffic is delivered again at the subscribers after a failure has been detected by the affected BFR.

9.2. Link Protection

We perform experiments for the evaluation of BIER-FRR with link protection. First, we explain the experimental setup. Afterwards, we report and discuss the results for all scenarios.

9.2.1. Setup for Link Protection

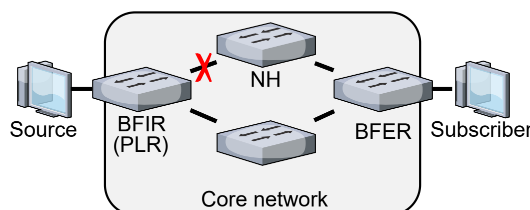


Figure 20: Two hosts the *Source* and the *Subscriber* are connected to a BIER network with IP as the routing underlay.

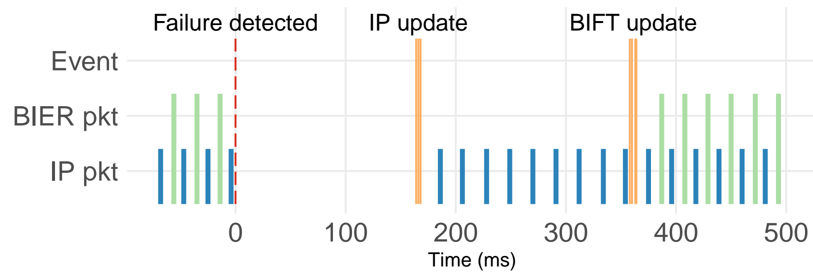
We emulate the testbed depicted in Figure 20 in Mininet. Two hosts the *Source* and the *Subscriber* are connected to a BIER/IP network. The host *Source* sends every 10 ms packets to the host *Subscriber* over the core network. Every other packet is sent by IP unicast and IPMC. The primary path carries packets from PLR via NH to $BFER$. We simulate the failure of the link between the PLR and the NH to interrupt packet delivery. We compare the time until the host *Subscriber* receives unicast and multicast traffic again, after the failure has been detected by the PLR. We perform experiments with and without IP-FRR and BIER-FRR with link protection.

9.2.2. Without IP-FRR and BIER-FRR

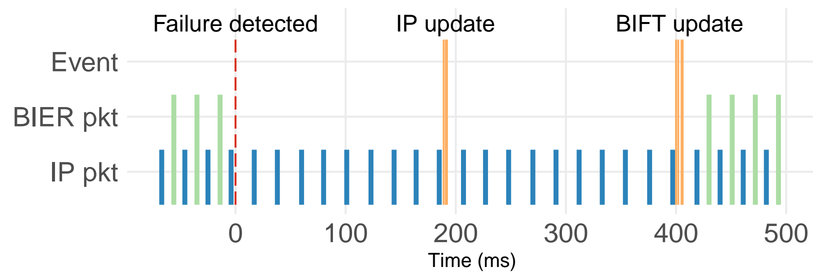
In the first experiment, failure recovery is based only on IP reconvergence and BIFT recomputation. Neither IP-FRR nor BIER-FRR are enabled. Figure 21(a) shows that the failure interrupts packet delivery at the *Subscriber*. Unicast reconvergence is completed after about 170 ms after failure detection. Updating the BIFT entries has finished only after about 370 ms in total. Unicast and multicast packets are received again by the *Subscriber* only after updated IP and BIER forwarding rules from the controller have been installed at the PLR.

9.2.3. With IP-FRR but without BIER-FRR

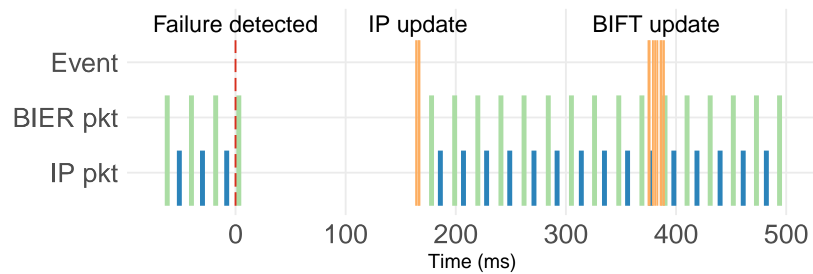
In the second experiment, IP-FRR is enabled but BIER-FRR remains disabled. Figure 21(b) shows that IP unicast traffic immediately benefits from



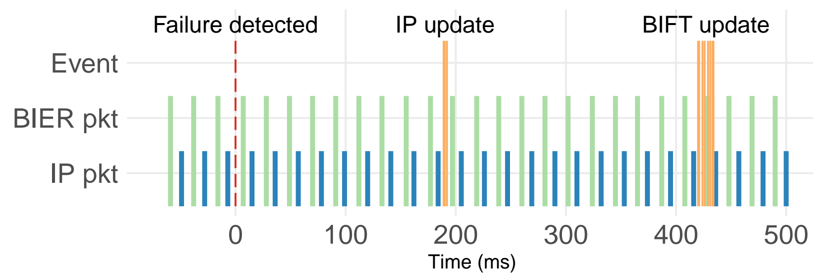
(a) Without IP-FRR and BIER-FRR.



(b) With IP-FRR but without BIER-FRR.



(c) Without IP-FRR but with BIER-FRR.



(d) With IP-FRR and BIER-FRR.

Figure 21: Reception time of packets in the link failure scenario.

IP-FRR when the *PLR* detects the failure. IP-FRR instantly reroutes packets and, therefore, IP unicast traffic is still delivered at the *Subscriber*. Both IP reconvergence and BIFT recompuration are finished slightly later compared to the previous scenario. The reason for the extended duration is that the global controller needs to compute new forwarding entries for IP-FRR during reconvergence, which is not needed if IP-FRR is disabled. After 200 ms, IP reconvergence has finished and the primary IP unicast forwarding entries have been updated. Multicast packets are delivered only after BIFT recomputation after about 400 ms.

9.2.4. Without IP-FRR but with BIER-FRR

In the third experiment, IP-FRR is disabled but BIER-FRR is enabled. Figure 21(c) shows that unicast traffic is delivered at the *Subscriber* when IP reconvergence has finished after about 170 ms. Due to BIER-FRR, BIER traffic benefits from the faster IP reconvergence, too. Multicast traffic is delivered after 170 ms as well, and not only after BIFT recomputation. The BIFT is updated only after about 400 ms in total which is slightly longer than in the scenario without BIER-FRR. Although conceptually the BIFT does not require modification for BIER-FRR with link protection, the match+action tables in the P4 implementation need backup entries that tunnel BIER packets in case of a failure. Therefore, the global controller has to compute new backup entries for BIER-FRR in addition to primary BIFT entries during the recomputation process. The slightly delayed BIFT recomputation is not a disadvantage for BIER traffic because BIER-FRR reroutes BIER packets until both primary and backup BIFT entries have been updated.

9.2.5. With IP-FRR and BIER-FRR

In the last experiment, IP-FRR and BIER-FRR are enabled. Figure 21(d) illustrates that both unicast and multicast traffic are delivered at the *Subscriber* without any delay despite of the failure. This is achieved by FRR mechanisms in both the routing underlay and the BIER layer. IP-FRR immediately restores connectivity for unicast traffic. BIER-FRR leverages the resilient routing underlay to immediately reroute BIER packets. IP reconvergence has finished after about 200 ms. BIFT recomputation finishes only after about 420 ms. In both cases the longer time is explained by the additional FRR entries the global controller has to compute during IP reconvergence and BIFT recomputation, respectively.

9.3. Node Protection

In this paragraph we evaluate BIER-FRR with node protection. First, we describe the experimental setup. Then, we report and discuss the evaluation results for all four scenarios.

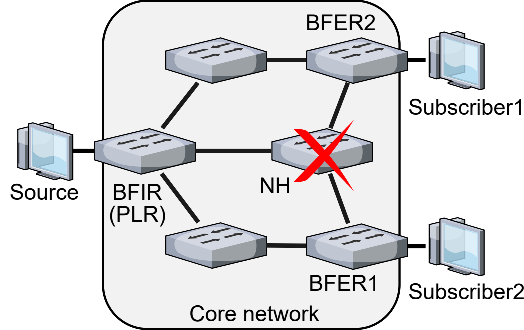


Figure 22: Three hosts, $Source$, $Subscriber1$ and $Subscriber2$ are connected to a BIER network with IP as the routing underlay.

9.3.1. Setup for Node Protection

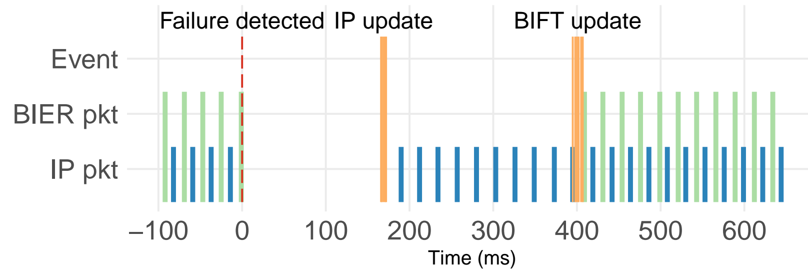
Figure 22 shows the topology we emulated in Mininet. The three hosts $Source$, $Subscriber1$, and $Subscriber2$ are connected to an BIER/IP network. The $Source$ alternately sends two IP unicast packets and one IP multicast packet with 10 ms in between. The unicast packets are sent to $Subscriber1$ and $Subscriber2$. The IPMC group of the the IPMC packet is subscribed by $Subscriber1$ and $Subscriber2$. On the primary path, packets are carried from the PLR via the NH to $BFER1$ and $BFER2$, respectively. We simulate the failure of the NH to interrupt packet delivery with a node failure. We evaluate the time until both the $Subscriber1$ and the $Subscriber2$ receive traffic again after the PLR detects the failure. We perform experiments with and without IP-FRR and BIER-FRR with node protection. We discuss the outcome and show figures only for $Subscriber1$ because results for $Subscriber2$ are very similar.

9.3.2. Without IP-FRR and BIER-FRR

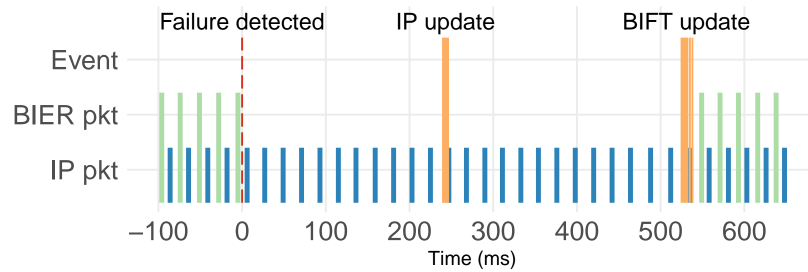
In the first scenario, the local controller at the PLR triggers only IP reconvergence and BIFT recomputation after failure detection. No FRR measures are enabled. Figure 23(a) shows that the $Subscriber1$ receives IP unicast traffic only after IP reconvergence which takes about 180 ms. $Subscriber1$ receives multicast traffic only after BIFT recomputation which takes about 400 ms. Both IP reconvergence and BIFT recomputation require slightly more time than in the link failure scenario because now the local controller reports a node failure which requires more rules to be recomputed.

9.3.3. With IP-FRR but without BIER-FRR

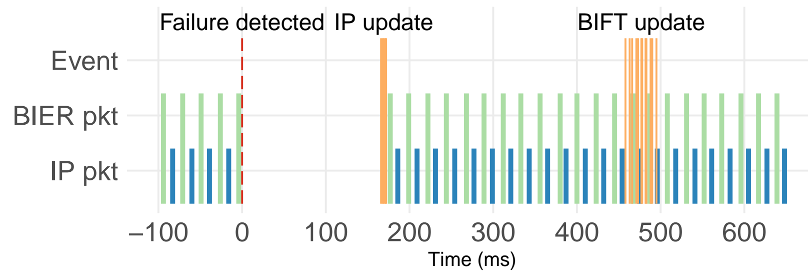
In the second scenario, IP-FRR is enabled but not BIER-FRR. Figure 23(b) shows that IP unicast traffic immediately benefits from IP-FRR. Traffic is delivered at the $Subscriber1$ without any delay despite of the failure. IP reconvergence requires about 240 ms. Multicast traffic is received by the $Subscriber1$ only after BIFT recomputation which has finished only after about 520 ms.



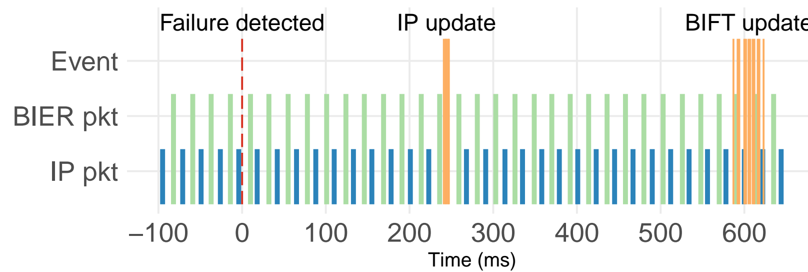
(a) Without IP-FRR and BIER-FRR.



(b) With IP-FRR but without BIER-FRR.



(c) Without IP-FRR but with BIER-FRR.



(d) With IP-FRR and BIER-FRR.

Figure 23: Reception time of packets in the node failure scenario.

Again, IP reconvergence and BIFT recomputation require slightly more time than without IP-FRR because because additional IP-FRR entries have to be computed.

9.3.4. Without IP-FRR but with BIER-FRR

In the third scenario, BIER-FRR is enabled but not IP-FRR. Figure 23(b) shows that both IP unicast and multicast traffic are received at the $Subscriber1$ only after IP reconvergence which takes about 170 ms. Afterwards, IP traffic is rerouted because of the updated forwarding entries. BIER traffic is rerouted after that time as well, because BIER-FRR leverages the updated routing underlay instead of requiring BIFT recomputation which has finished only after about 500 ms.

9.3.5. With IP-FRR and BIER-FRR

In the last scenario, both IP-FRR and BIER-FRR are enabled. Figure 23(d) shows that both IP unicast and multicast traffic are received by the $Subscriber1$ without any delay despite of the failure. IP-FRR reroutes IP unicast traffic as soon as the failure is detected by the PLR. Similarly, BIER-FRR reroutes BIER traffic immediately, too. Therefore, BIER traffic benefits from the resilience of the routing underlay to forward BIER traffic although the NH failed and BIFT recomputation has not finished, yet. IP reconvergence takes about 240 ms. BIFT recomputation finished only after 600 ms.

10. Conclusion

BIER is a novel, domain-based, scalable multicast transport mechanism for IP networks that does not require state per IP multicast (IPMC) group in core nodes. Only ingress nodes of a BIER domain maintain group-specific information and push a BIER header on multicast traffic for simplified forwarding within the BIER domain. Bit-forwarding routers (BFRs) leverage a bit index forwarding table (BIFT) for forwarding decisions. Its entries are derived from the interior gateway protocol (IGP), the so-called routing underlay. In case of a failure, the BIFT entries are recomputed only after IP reconvergence. Therefore, BIER traffic encounters rather long outages after link or node failures and cannot profit from fast reroute (FRR) mechanisms in the IP routing underlay.

In this work, we proposed BIER-FRR to shorten the time until BIER traffic is delivered again after a failure. BIER-FRR deviates BIER traffic around the failure via unicast tunnels through the routing underlay. Therefore, BIER benefits from fast reconvergence or FRR mechanisms of the routing underlay to deliver BIER traffic as soon as connectivity for unicast traffic has been restored in the routing underlay. BIER-FRR has a link and a node protection mode. Link protection is simple but cannot protect against node failures. To that end, BIER-FRR offers a node protection mode which requires extensions to the BIFT structure.

As BIER defines new headers and forwarding behavior, it cannot be configured on standard networking gears. Therefore, a second contribution of

this paper is a prototype implementation of BIER and BIER-FRR on a P4-programmable switch based on $P4_{16}$. It works without extern functions or other extensions such as local agents that impede portability. The switch offers an API for interaction with controllers. A local controller takes care of local tasks such as MAC learning and failure detection. A global controller configures other match+action tables that pertain to forwarding decisions. A predecessor of this prototype without BIER-FRR and based on $P4_{14}$ has been presented as a demo in [5]. The novel BIER prototype including BIER-FRR demonstrates that P4 facilitates implementation of rather complex forwarding behavior.

We deployed our prototype on a virtualized testbed based on Mininet and the software switch BMv2. Our experiments confirm that BIER-FRR significantly reduces the time until multicast traffic is received again by subscribers after link or node failures. Without BIER-FRR, multicast packets arrive at the subscriber only after reconvergence of the routing underlay and BIFT recomputation. With BIER-FRR, multicast traffic is delivered again as soon as connectivity in the routing underlay is restored, which is particularly fast if the routing underlay applies FRR methods.

Acknowledgment

The authors thank Wolfgang Braun and Toerless Eckert for valuable input and stimulating discussions.

References

[1] D. Merling, M. Menth, et al., An Overview of Bit Index Explicit Replication (BIER), IETFJournal (Mar. 2018).

[2] I. Wijnands, E. Rosen, et al., RFC8279: Multicast Using Bit Index Explicit Replication (BIER), https://tools.ietf.org/html/rfc8279 (Nov. 2017).

[3] M. Shand, S. Bryant, IP Fast Reroute Framework, https://tools.ietf.org/html/rfc5714 (Jan. 2010).

[4] D. Merling, M. Menth, BIER Fast Reroute, https://datatracker.ietf.org/doc/draft-merling-bier-frr/ (Mar. 2019).

[5] W. Braun, J. Hartmann, et al., Demo: Scalable and Reliable Software-Defined Multicast with BIER and P4, IFIP/IEEE International Symposium on Integrated Network Management (IM) (May 2017).

[6] The P4 Language Consortium, The P4 Language Specification Version 1.0.5, https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf (Nov. 2018).

[7] The P4 Language Consortium, The P4 Language Specification Version 1.1.0, https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf (Nov. 2018).

[8] H. Holbrook, B. Cain, et al., Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast, https://tools.ietf.org/html/rfc4604 (Aug. 2006).

[9] T. Speakman, J. Crowcroft, et al., PGM Reliable Transport Protocol Specification, https://tools.ietf.org/html/rfc3208 (Dec. 2001).

[10] G. Rétvári, J. Tapolcai, et al., IP fast ReRoute: Loop Free Alternates revisited, IEEE Conference on Computer Communications (Apr. 2011).

[11] D. Katz, D. Ward, et al., Bidirectional Forwarding Detection (BFD), https://tools.ietf.org/html/rfc5880 (Jun. 2010).

[12] D. Merling, W. Braun, et al., Efficient Data Plane Protection for SDN, IEEE Conference on Network Softwarization and Workshops (Jun. 2018).

[13] M. Menth, M. Hartmann, et al., Loop-Free Alternates and Not-Via Addresses: A Proper Combination for IP Fast Reroute?, Computer Networks 54 (Jun. 2010).

[14] A. Raj, O. Ibe, et al., A survey of IP and multiprotocol label switchingfast reroute schemes, Computer Networks 51 (Jun. 2007).

[15] V. S. Pal, Y. R. Devi, et al., A Survey on IP Fast Rerouting Schemes using Backup Topology, International Journal of Advanced Research inComputer Science and Software Engineering 3 (Apr. 2003).

[16] S. Bryant, S. Previdi, et al., A Framework for IP and MPLS Fast Reroute Using Not-Via Addresses, https://tools.ietf.org/html/rfc6981 (Aug. 2013).

[17] L. Csikor, G. Rétvári, et al., IP fast reroute with remote Loop-Free Alternates: The unit link cost case, International Congress on Ultra Modern Telecommunications and Control Systems (Feb. 2012).

[18] S. Islam, N. Muslim, et al., A Survey on Multicasting in Software-Defined Networking, IEEE Communications Surveys Tutorials 20 (Nov. 2018).

[19] Z. Al-Saeed, I. Ahmada, et al., Multicasting in Software Defined Networks: A Comprehensive Survey, Journal of Network and Computer Applications 104 (Feb. 2018).

[20] J. Rückert, J. Blendin, et al., Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks, Journal of Network and Systems Management 23 (Apr. 2015).

[21] J. Rückert, J. Blendin, et al., Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm, IEEE Transactions on Network and Service Management 13 (Sep. 2016).

[22] L. H. Huang, H.-J. Hung, et al., Scalable and Bandwidth-Efficient Multicast for Software-Defined Networks, IEEE Global Communications Conference (Dec. 2014).

[23] S. Zhou, H. Wang, et al., Cost-Efficient and Scalable Multicast Tree in Software Defined Networking, Algorithms and Architectures for Parallel Processing (Dec. 2015).

[24] J.-R. Jiang, S.-Y. Chen, Constructing Multiple Steiner Trees for Software-Defined Networking Multicast, Proceedings of the 11th International Conference on Future Internet Technologies (Jun. 2016).

[25] Y.-D. Lin, Y.-C. Lai, et al., Scalable Multicasting with Multiple Shared Trees in Software Defined Networking, Journal of Network and Computer Applications 78 (Jan. 2017).

[26] Z. Hu, D. Guo, et al., Multicast Routing with Uncertain Sources in Software-Defined Network, IEEE/ACM International Symposium on Quality of Service (Jun. 2016).

[27] B. Ren, D. Guo, et al., The Packing Problem of Uncertain Multicasts, Concurrency and Computation: Practice and Experience 29 (August 2017).

[28] A. Iyer, P. Kumar, et al., Avalanche: Data Center Multicast using Software Defined Networking, International Conference on Communication Systems and Networks (Jan 2014).

[29] W. Cui, C. Qian, et al., Scalable and Load-Balanced Data Center Multicast, IEEE Global Communications Conference (Dec 2015).

[30] S. H. Shen, L.-H. Huang, et al., Reliable Multicast Routing for Software-Defined Networks, IEEE Conference on Computer Communications (April 2015).

[31] M. Popovic, R. Khalili, et al., Performance Comparison of Node-Redundant Multicast Distribution Trees in SDN Networks, International Conference on Networked Systems (Apr. 2017).

[32] T. Humernbrum, B. Hagedorn, et al., Towards Efficient Multicast Communication in Software-Defined Networks, IEEE International Conference on Distributed Computing Systems Workshops (Jun. 2016).

[33] D. Kotani, K. Suzuki, et al., A Multicast Tree Management Method Supporting Fast Failure Recovery and Dynamic Group Membership Changes in OpenFlow Networks, Journal of Information Processing 24 (2016).

[34] T. Pfeiffenberger, J. L. Du, et al., Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow, International Conference on New Technologies, Mobility and Security (Jul. 2015).

[35] W. K. Jia, L.-C. Wang, et al., A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks, IEEE Journal on Selected Areas in Communications 31 (Dec. 2013).

[36] M. J. Reed, M. Al-Naday, et al., Stateless Multicast Switching in Software Defined Networks, IEEE International Conference on Communications (May 2016).

[37] A. Giorgetti, A. Sgambelluri, et al., First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting, European Conference on Networks and Communications (Jun. 2017).

[38] A. Giorgetti, A. Sgambelluri, et al., Bit Index Explicit Replication (BIER) Multicasting in Transport Networks, International Conference on Optical Network Design and Modeling (May 2017).

[39] T. Eckert, G. Cauchie, et al., Traffic Engineering for Bit Index Explicit Replication BIER-TE, http://tools.ietf.org/html/draft-eckert-bier-te-arch (Nov. 2017).

[40] W. Braun, M. Albert, et al., Performance Comparison of Resilience Mechanisms for Stateless Multicast Using BIER, IFIP/IEEE International Symposium on Integrated Network Management (May 2017).

[41] Q. Xiong, G. Mirsky, et al., The Resilience for BIER, https://datatracker.ietf.org/doc/draft-xiong-bier-resilience/ (Mar. 2019).

[42] Q. Xiong, G. Mirsky, et al., BIER BFD, https://datatracker.ietf.org/doc/draft-hu-bier-bfd/ (Mar. 2019).

[43] D. Merling, S. Lindner, et al., Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast, International Conference on Software Defined Systems (Apr. 2020).

[44] P. Bosshart, D. Daly, et al., P4: Programming Protocol-Independent Packet Processors, ACM SIGCOMM Computer Communication Review 44 (Jul. 2014).

[45] M. Schmidt, F. Hauser, et al., LoCoSDN: A Local Controller for Operation of OFSwitches in non-SDN Networks, Software Defined System (Apr. 2018).

[46] F. Hauser, M. Schmidt, et al., P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection with MACsec in P4-Based SDN, IEEE Access (Mar. 2020).

[47] F. Hauser, M. Häberle, et al., P4-IPsec: Implementation of IPsec Gateways in P4 with SDN Control for Host-to-Site Scenarios, ArXiv (Jul. 2019).

[48] B. Lantz, B. Heller, et al., A Network in a Laptop: Rapid Prototyping for Software-defined Networks, ACM SIGCOMM HotNets Workshop (Oct. 2010).

[49] p4lang, behavioral-model, https://github.com/p4lang/behavioral-model (Mar. 2019).

1.6 Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4

IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received February 4, 2021, accepted February 19, 2021, date of publication February 24, 2021, date of current version March 4, 2021.
Digital Object Identifier 10.1109/ACCESS.2021.3061763

Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4

DANIEL MERLING, STEFFEN LINDNER, AND MICHAEL MENTH, (Senior Member, IEEE)
Chair of Communication Networks, University of Tuebingen, 72076 Tübingen, Germany
Corresponding author: Daniel Merling (daniel.merling@uni-tuebingen.de)

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under Grant ME2727/1-2. The authors alone are responsible for the content of this paper.

ABSTRACT Traditional IP multicast (IPMC) maintains state per IPMC group in core devices to distribute one-to-many traffic along tree-like structures through the network. This limits its scalability because whenever subscribers of IPMC groups change, forwarding state in the core network needs to be updated. Bit Index Explicit Replication (BIER) has been proposed by the IETF for efficient transport of IPMC traffic without the need of IPMC-group-dependent state in core devices. However, legacy devices do not offer the required features to implement BIER. P4 is a programming language which follows the software-defined networking (SDN) paradigm. It provides a programmable data plane by programming the packet processing pipeline of P4 devices. The contribution of this article is threefold. First, we provide a hardware-based prototype of BIER and BIER fast reroute (BIER-FRR) which leverages packet recirculation. Our target is the P4-programmable high-performance switching ASIC Tofino; the source code is publicly available. Second, we perform an experimental evaluation, with regard to failover time and throughput, which shows that up to 100 Gb/s throughput can be obtained and that failures affect BIER forwarding for less than 1 ms. However, throughput can decrease if switch-internal packet loss occurs due to missing recirculation capacity. As a remedy, we add more recirculation capacity by turning physical ports into loopback mode. To quantify the problem, we derive a prediction model for reduced throughput whose results are in good accordance with measured values. Third, we provide a provisioning rule for recirculation ports, that is applicable to general P4 programs, to avoid switch-internal packet loss due to packet recirculation. In a case study we show that BIER requires only a few such ports under realistic mixes of unicast and multicast traffic.

INDEX TERMS Software-defined networking, P4, bit index explicit replication, multicast, resilience, scalability.

I. INTRODUCTION

IP multicast (IPMC) has been proposed to efficiently distribute one-to-many traffic, e.g. for IPTV, multicast VPN, commercial stock exchange, video services, public surveillance data distribution, emergency services, telemetry, or content-delivery networks, by forwarding only one packet per link. IPMC traffic is organized in IPMC groups which are subscribed by hosts. Figure 1 shows the concept of IPMC. IPMC traffic is forwarded on IPMC-group-specific distribution trees from the source to all subscribed hosts. To that end, core routers maintain forwarding state for each IPMC group to determine the next-hops (NHs) of an IPMC packet. Scalability issues are threefold. First, a significant

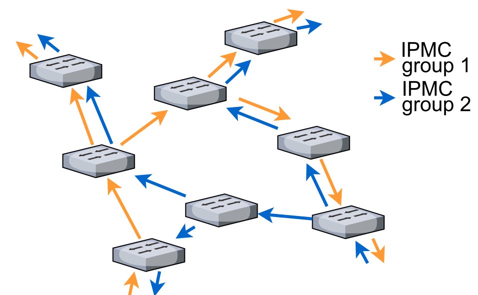


Figure 1. Two multicast distribution trees.

amount of storage is required to keep extensive forwarding state. Second, when subscribers of an IPMC group change, the distribution tree needs to be updated by signaling the changes to core devices. Third, the distribution trees have to

The associate editor coordinating the review of this manuscript and approving it for publication was Martin Reisslein.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

be updated when the topology changes or a failure is detected. Therefore, traditional IPMC comes with significant management and state overhead. As a result, traditional IPMC is often avoided and multicast is implemented on the application layer. Thereby, one-to-many traffic is carried via network layer unicast, which is not efficient.

The IETF proposed Bit Index Explicit Replication (BIER) [1] for efficient transport of IPMC traffic. BIER introduces a BIER domain where core routers do not need to maintain IPMC-group-dependent state. Upon entering the BIER domain, IPMC packets are equipped with a BIER header which specifies all destinations of the packet within the BIER domain. The BIER packets are forwarded through the BIER domain towards their destinations on paths from the Interior Gateway Protocol (IGP), which we call 'routing underlay' in the following. Thereby, only one packet is forwarded per link. When the BIER packets leave the BIER domain, the BIER header is removed.

Unicast and BIER traffic may be affected by failures. IP-Unicast traffic is often protected by fast reroute (FRR) mechanisms for IP (IP-FRR). IP-FRR leverages precomputed backup entries to quickly reroute a packet on a backup path when the primary NH is unreachable. Tunnel-based BIER-FRR [2] is used to protect BIER traffic by tunneling BIER packets through the routing underlay. The tunnel may be also affected by a failure, but FRR or timely updates of the forwarding information base (FIB) in the routing underlay quickly restore connectivity. However, BIER is not supported by legacy devices and there is no dedicated BIER hardware available. P4 [3] is a programming language that follows the software-defined networking (SDN) paradigm for programming protocol-independent packet processors. P4 allows developers to write high-level programs to define the packet processing pipeline of programmable network devices. A target-specific compiler translates the P4 program for execution on a particular device. With the P4-programmable data plane new protocols can be implemented and deployed in short time.

In previous work [2], [4] we implemented BIER and tunnel-based BIER-FRR for the P4 software switch bmv2 [5]. However, the developers of the bmv2 clarify that the 'BMv2 is not meant to be a production-grade software switch' [5] and is, therefore, only a 'tool for developing, testing and debugging P4 data planes' [5]. Thus, it remains unclear whether BIER and BIER-FRR forwarding is simple enough to be implemented also on P4-capable hardware platforms which entail functional and runtime constraints to achieve high-speed forwarding.

The contribution of this article is threefold. First, we provide a new prototype for BIER and BIER-FRR on the P4-programmable switching ASIC Tofino [6] which is used in the Edgecore Wedge 100BF-32X [7], a 32 100 Gb/s port high-performance P4 switch, and make our code publicly available.

Second, we conduct an experimental performance study with regard to failover time and throughput. The evaluations show that connectivity can be restored within less than 1 ms and that a throughput of up to 100 Gb/s can be obtained. However, we observe reduced throughput under certain conditions and conjecture that this results from switch-internal packet loss due to missing recirculation capacity. We add more recirculation capacity by turning physical ports into loopback mode to avoid switch-internal packet loss in case of recirculation. To quantify the problem, we derive a prediction model for BIER throughput whose results are in good accordance with measured values.

Third, we propose a provisioning rule for recirculation ports to avoid switch-internal packet loss due to packet recirculation. It is applicable to general P4 programs and helps to avoid throughput reduction on outgoing links. Finally, we utilize the provisioning model to show in a case study that only a few ports in loopback mode suffice to avoid internal packet loss with BIER under realistic mixes of unicast and multicast traffic.

The paper is structured as follows. In Section II we describe related work. Section III contains a primer on BIER and tunnel-based BIER-FRR. Afterwards, we give an overview on P4 in Section IV and explain important properties. In Section V, we briefly describe the P4 implementation of BIER and tunnel-based BIER-FRR for the Tofino. Section VI contains our evaluation and the model for throughput prediction of BIER. In Section VII we present a model to provision recirculation ports. We conclude the paper in Section VIII.

II. RELATED WORK

First, we describe related work for SDN-based multicast in general. Then, we review work for BIER-based multicast. Finally, we present P4 projects that are based on packet recirculation.

A. SDN-BASED MULTICAST

Elmo [8] increases scalability of traditional IPMC in data center environments by leveraging characteristics of data center networks, in particular symmetric topologies and short paths. By encoding multicast group information in the packet header, this information is no longer stored in forwarding devices. This significantly reduces the dynamic state that needs to be maintained by core nodes.

Two surveys [9], [10] provide a comprehensive overview of SDN-based multicast. They review the development of traditional multicast and different aspects of SDN-based multicast, e.g., building of distribution trees, group management, and approaches to improve the efficiency of multicast. Most of the papers in the surveys discuss multicast mechanisms that are based on explicit IPMC-group-dependent state in core devices. The downsides of those traditional IPMC approaches have been discussed in Section I. We still discuss some papers on IPMC due to their efforts to make traditional IPMC more efficient. The papers often focus on intelligent tree building mechanisms that reduce the state, or efficient signaling techniques when IPMC groups or the topology changes. The surveys also consider works that utilize SDN to

improve multicast. They are related as our approach also takes an SDN approach. Therefore, we present some representative examples from that area.

1) OPTIMIZATION OF MULTICAST TREES

Rückert *et al.* propose Software-Defined Multicast (SDM) [11]. SDM is an OpenFlow-based platform that provides well-managed multicast for over-the-top and overlay-based live streaming services tailored for P2P-based video stream delivery. The authors extend SDM in [12] with traffic engineering capabilities. In [13] the authors propose address translation from the multicast address to the unicast address of receivers at the last multicast hop in OpenFlow switches. This reduces the number of IPMC-group-dependent forwarding entries in some nodes.

Steiner trees are often used to build multicast distribution trees [14]. Several papers modify the original Steiner-tree problem to build distribution trees with minimal cost [15], number of edges [16], number of branch nodes [17], delay [18], or for optimal position of the multicast source [19].

The authors of [20] implement a multicast platform in OpenFlow with a reduced number of forwarding entries. It is based on multiple shared trees between different IPMC groups. The Avalanche Routing Algorithm (AvRA) [21] considers properties of the topology of data center networks to build trees with optimal utilization of network links. Dual-Structure Multicast (DuSM) [22] leverages different forwarding structures for high-bandwidth and low-bandwidth flows. This improves scalability and link utilization of SDN-based data centers. Jia *et al.* [23] present a way to efficiently organize forwarding entries based on prime numbers and the Chinese remainder theorem. This reduces the required state in forwarding devices and allows more efficient implementation. In [24] the authors propose a SDN-based multicast switching system that leverages bloom filters to reduce the number of TCAM-entries.

2) RESILIENCE FOR TRADITIONAL MULTICAST

Shen *et al.* [25] modify Steiner trees to include recovery nodes in the multicast distribution tree. The recovery nodes cache IPMC traffic temporarily and resend it after reconvergence when the destination notified the recovery point because it did not get all packets due to a failure. The authors of [26] evaluate several algorithms that generate node-redundant multicast distribution trees. They analyse the number of forwarding entries and the effect of node failures. In [27] the authors propose to deploy primary and backup multicast trees in SDN networks. The header of multicast packets contains an ID that identifies the distribution tree on which the packet is forwarded. When a failure is detected, the controller reconfigures affected sources to send packets along a working backup tree. Pfeiffenberger *et al.* [28] propose a similar method. Each node that is part of a distribution tree is the root of a backup tree that does not contain the unreachable NH but all downstream destinations of the primary distribution tree. When a node cannot forward a packet, it reroutes the packet on a backup tree by switching an VLAN tag in the packet header.

B. BIER-BASED MULTICAST

In this subsection we discuss work directly related to BIER. First, we define our work in contrast to other implementations. Then, we describe evaluations and extensions for BIER.

1) IMPLEMENTATIONS

We started with an implementation of BIER for the software switch bmv2 using $P4_{14}$. The protoype was documented at high level in a 2-page demo paper [4]. We then developed BIER-FRR and implemented a prototype for BIER and BIER-FRR on the software switch bmv2 using the newer variant $P4_{16}$ in [2]. That work demonstrated that the P4 language is expressive enough to implement also complex forwarding mechanisms and introduced a hierarchical controller hierarchy to quickly trigger FRR actions. The study compared restoration times for various failure cases and protection schemes at light load conditions of a few packets per second. Throughput measurements were not conducted as the bmv2 software switch is only a ‘tool for developing, testing and debugging P4 data planes’ [5] with low throughput (900 Mb/s) [29] and not for application in real networks. In contrast, this paper shows that BIER and BIER FRR can be implemented also on high-performance P4-programmable hardware, i.e., the switching ASIC Tofino, which entails additional functional and runtime constraints for implementations to achieve high throughput. Experimental measurement studies in a 100 Gb/s hardware testbed reveal performance challenges due to recirculations. As this is a general problem for some P4 programs, we derive recommendations to cope with them and validate them in our hardware testbed.

We know only a single BIER implementation by other authors which is based on OpenFlow and presented in [30], [31]. Their approach suffers from two major shortcomings. First, the BIER bit string is encoded in a MPLS header which is the only way to encode arbitrary bit strings in OpenFlow. This limits the bit string length, and thus the number of receivers, to 20 which is the length of an MPLS label. Second, the implementation performs an exact match on the bitstring. If a subscriber changes, the match does not work anymore and a local BIER agent that is not part of the OpenFlow protocol needs to process the packet. Therefore, we consider this project only as an early BIER-based prototype for OpenFlow and not as a production-ready BIER implementation.

2) EVALUATIONS AND EXTENSIONS OF BIER-BASED MULTICAST

The authors of [32] perform a simulation-based evaluation of BIER. They find that on metrics like delivery ratios and retransmissions BIER performs as well as traditional IPMC but has better link usage and no per-flow or per-group state in core devices.

Eckert *et al.* [33] propose an extension for BIER that allows for traffic engineering (BIER-TE). In addition to the egress nodes, the BIER header encodes the distribution tree of a packet. In [34] the authors propose 1 + 1 protection for BIER-TE. The traffic is transported on two disjoint distribution trees, which delivers the traffic even if one tree is interrupted by a failure.

C. PACKET RECIRCULATION IN P4

Hauser *et al.* [35] show in their P4 survey that packet recirculation is not used only in this BIER implementation but also in other P4 projects. In [36] the authors implement a congestion control mechanism in P4 and leverage packet recirculation to create notification packets, update their header fields, and send them to appropriate monitoring nodes. The authors of [37] present a content-based publish/subscribe mechanism in P4 where they introduce a new header stack that requires packet recirculation for processing. Uddin *et al.* [38] implement multi-protocol edge switching for IoT based on P4. Packet recirculation is used to process packets a second time after they have been decrypted.

III. BIT INDEX EXPLICIT REPLICATION (BIER)

In this Section we explain BIER. First, we give an overview. Then we describe the BIER forwarding table and how BIER packets are processed. Afterwards, we show a forwarding example. Finally, we review tunnel-based BIER-FRR.

A. BIER OVERVIEW

First, we introduce the BIER domain. Then, we present the layered BIER architecture followed by the BIER header. Finally, we describe BIER forwarding.

1) BIER DOMAIN

Figure 2 shows the concept of the BIER domain. When bit-forwarding ingress routers (BFIRs) receive an IPMC packet they push a BIER header onto it and forward the packet into the BIER domain. The BIER header identifies all destinations of the BIER packet within the BIER domain, i.e., bit-forwarding egress routers (BFERs). Bit-forwarding routers (BFRs) forward the BIER packets to all BFERs indicated in its BIER header. Thereby, packets are replicated and forwarded to multiple next-hops (NHs) but only one packet is sent over any involved link. The paths towards the destinations are provided by the Interior Gateway Protocol (IGP), i.e., the routing underlay. Therefore, from a specific BFIR to a specific BFER, the BIER packet follows the same path as unicast traffic. Finally, BFERs remove the BIER header.

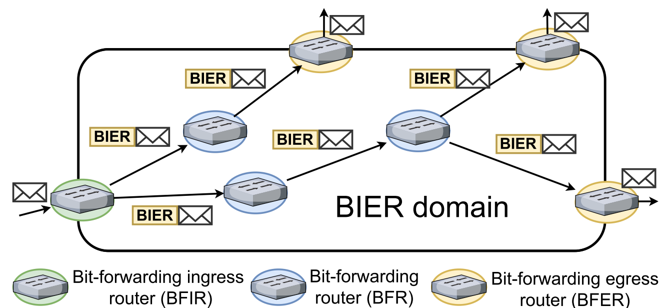


Figure 2. The concept of the BIER domain [39].

2) THE LAYERED BIER ARCHITECTURE

The BIER architecture consists of three components. The IPMC layer, the BIER layer and the routing underlay. Figure 3 shows the three layers, their composition, and interaction. The IPMC layer contains the sources and subscribers of IPMC traffic. The BIER layer acts as a transport layer for IPMC traffic. It consists of the BIER domain which is connected to the IPMC layer at the BFIRs, and BFERs. Therefore, the BIER layer acts as a point-to-multipoint tunnel from an IPMC source to multiple subscribers. The routing underlay refers to the IGP which provides the paths to all destinations within the network.

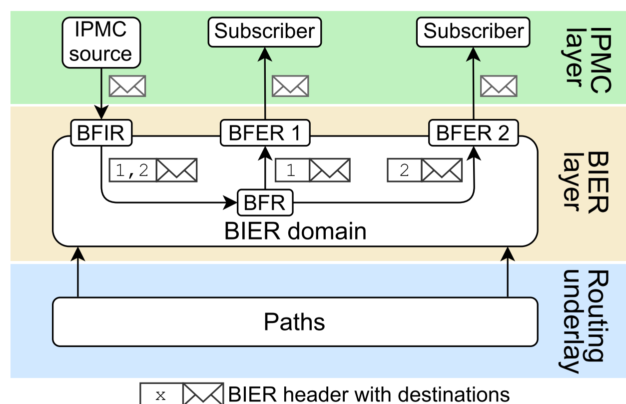


Figure 3. IPMC packets are transmitted over a layered BIER architecture; the paths are defined by the information from the routing underlay [39].

3) BIER HEADER

The BIER header contains a bit string to indicate the destinations of a BIER packet. To that end, each BFER is assigned an unique number that corresponds to a bit position in that bit string, starting by 1 for the least-significant bit. If a BFER should receive a copy of the IPMC packet, its bit is activated in the bit string in the BIER header of the packet. To facilitate readability we refer to the bit string in the BIER header of a BIER packet with the term 'BitString'.

4) BIER FORWARDING

A BFR forwards a packet copy to any neighbor over which at least one destination of the packet indicated by its BitString is reached according to the paths from the routing underlay. Before a packet is forwarded to a specific NH, the BFR clears all bits that correspond to BFERs that are reached via other NHs from the BitString of that packet. This prevents duplicates at the BFERs.

B. BIFT STRUCTURE

BFRs use the Bit Index Forwarding Table (BIFT) to determine the NHs of a BIER packet. Table 1 shows the BIFT of BFR 1 from Figure 4. For each BFER there is one entry in the BIFT. Entries of the BIFT consist of a NH, and a so-called F-BM. The F-BM is a bit string similar to the BitString. It records which BFERs have the same NH. In the F-BM of an BIFT entry the bits of BFERs are activated which are reached over the NH of that entry. Therefore, BFERs with the same NH have the same F-BM. BFRs use the F-BM to clear bits from the BitString of a packet before it is forwarded to a NH.

Table 1. BIFT of BFR 1 in the example of Figure 4 [39].

BFER	NH	F-BM
1	-	-
2	2	1010
3	3	0100
4	2	1010

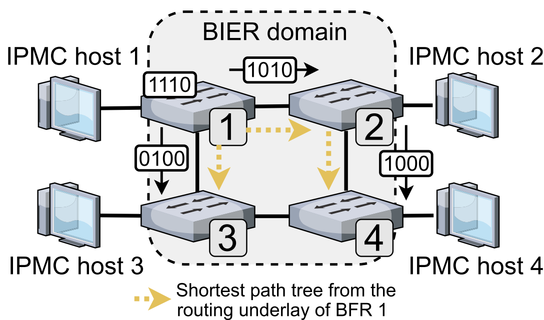


Figure 4. Example of a BIER topology and BitStrings of forwarded BIER packets [39].

C. BIER PACKET PROCESSING

When a BFR receives a BIER packet, it first stores the BitString of the packet in a separate bit string to account to which BFERs a packet has to be sent. In the following, we refer to that bit string with the term 'remaining bits'. The following procedure is repeated, until the remaining bits contain no activated bits anymore [1].

The BFR determines the least-significant activated bit in the remaining bits. The BFER that corresponds to that bit is used for a lookup in the BIFT. If a matching entry is found, it results in a NH *nh* and the F-BM *fbm* and the BFR creates a copy of the BIER packet. The BFR uses *fbm* to clear bits from the BitString of the packet copy. To that end, the BFR performs a bitwise AND operation of *fbm* and the BitString of the packet copy and writes the result into the BitString of the packet copy. This procedure is called applying the F-BM. It leaves only bits of BFERs in the BitString active that are reached over *nh*. The packet copy is then forwarded to *nh*. Afterwards, the bits of BFERs to which a packets has just been sent are cleared from the remaining bits. To that end, the BFR performs a bitwise AND operation of the bitwise complement of *fbm* with the remaining bits. The result is then stored in the remaining bits.

D. BIER FORWARDING EXAMPLE

Figure 4 shows a topology with four BIER devices where each is BFIR, BFR, and BFER. Table 1 shows the BIFT of BFR 1.

BFR 1 receives an IPMC packet from IPMC host 1 which should be distributed to all other IPMC hosts. Therefore, BFIR 1 pushes a BIER header with the BitString `1110` to the IPMC packet.

Then, BFR 1 determines the least-significant activated bit in the BIER header which corresponds to BFER 2. This BFER is used for lookup in the BIFT, which results in the F-BM `1010` and the NH BFR 2. BFR 1 creates a packet copy and applies the F-BM to its BitString. Then, the packet copy with the BitString `1010` is forwarded to BFR 2. Finally, the activated bits of the F-BM are cleared from the remaining bits which leaves the bit string `0100`.

This leaves only one bit active which identifies BFER 3. After the F-BM `0100` is applied to the BitString of a packet copy, it is forwarded to BFR 3 with the BitString `0100`. After clearing the bits of the F-BM from the remaining bits, processing stops because no active bits remain.

E. TUNNEL-BASED BIER-FRR

Tunnel-based BIER-FRR is used to deliver BIER traffic even when NHs are unreachable due to link or node failures. When a BFR detects that a NH is unreachable, e.g., by loss-of-carrier, loss-of-light, or a bidirectional forwarding detection (BFD[1]) [40] for BIER [41], it becomes the point of local repair (PLR) by tunneling the BIER packet through the routing underlay to nodes downstream in the BIER distribution tree. The tunnel may be affected by the failure, too. However, FRR mechanisms or timely updates of the FIB in the routing underlay restore connectivity for unicast traffic faster than for BIER traffic because recomputation of BIER entries can start only after the FIB of the routing underlay has been updated. Tunnel-based BIER-FRR can be configured either for link protection or node protection. BIER-FRR with link protection tunnels the BIER packet to the NH where the tunnel header is removed and the BIER header is processed again. BIER-FRR with node protection tunnels copies of the BIER packets to all next-next-hops (NNHs) in the distribution tree.

IV. INTRODUCTION TO P4

In this section we briefly review fundamentals of P4 [3]. First, we give an short overview of the P4 processing pipeline. Afterwards, we explain packet cloning and packet recirculation and point out important properties.

A. P4 PIPELINE

In this subsection we review the P4 processing pipeline. We explain its composition, transient and persistent memory, match + action tables, control blocks, packet cloning

[1]When a BFR is established between two nodes, they periodically exchange notifications about their status.

and packet recirculation. Figure 5 shows the concept of the P4 processing pipeline.

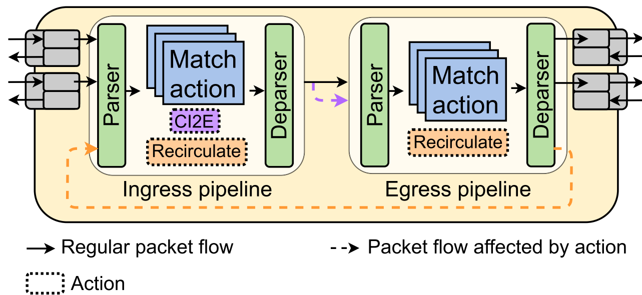


Figure 5. P4 processing pipeline.

1) COMPOSITION

The P4 pipeline consists of an ingress pipeline and an egress pipeline. They process packets in a similar fashion, i.e., both contain a parser, a match + action pipeline, and a deparser. When a packet arrives at the switch, it is first processed by the ingress pipeline. The header fields of the packet are parsed and carried along with the packet through the ingress pipeline. The parser is followed by a match + action pipeline which consists of a sequence of conditional statements, table matches, and primitive operations. Afterwards, the packet is deparsed and sent to the egress pipeline for further processing. Finally, the packet is sent through the specified egress port which has to be set in the ingress pipeline and cannot be changed in the egress pipeline.

The P4 program defines the parser and the deparser, which allows the use of custom packet headers. In addition, the P4 program describes the control flow of the match + action pipeline in the ingress pipeline and egress pipeline, respectively.

2) CONTROL BLOCKS

Both the ingress and egress pipeline can be divided into so-called control blocks for structuring. Control blocks are used to clearly separate functionality for different protocols like IP, BIER, and Ethernet, i.e., the IP control block contains Match + Action Tables (MATs) and operations that are applied only to IP packets, etc. In this paper we focus only on the BIER control block.

3) Match+Action TABLES (MATs)

MATs execute packet-dependent actions by matching packet header fields against MAT entries. To that end, an entry contains one or more match fields, and an action set. When a packet is matched against a MAT, the match fields of the entries are compared with specified header fields of the packet. An action set consists of one or more actions, e.g., reading or writing a header field, mathematical operations, setting the egress port of the packet, etc. It is not possible to match a packet on the same MAT multiple times.

B. PACKET CLONING

The operation clone-ingress-to-egress (CI2E) allows packet replication in P4. It can be called only in the ingress pipeline. At the end of the ingress pipeline, a copy of the packet is created. However, the packet copy resembles the packet that has been parsed in the beginning of the ingress pipeline, i.e., the header changes performed during processing in the ingress pipeline are reverted. This is illustrated in Figure 6.

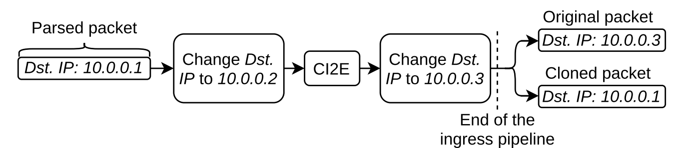


Figure 6. An example of the clone-ingress-to-egress (CI2E) operation [39].

If an egress port has been provided as a parameter, the egress port of the clone is set to that port. Both the original and cloned packet are processed independently in the egress pipeline. The cloned packet carries a flag to identify it as a clone.

C. PACKET RECIRCULATION

In this subsectin we explain the packet recirculation operation. First, we explain its working. Afterwards, we introduce the term recirculation capacity.

1) FUNCTIONALITY

P4 allows to recirculate a packet for processing it by the pipeline a second time. We use this feature to implement the iterative packet processing of BIER as described in Section III-C as P4 offers no other possibility to implement processing loops.

P4 leverages a switch-intern recirculation port for packet recirculation. When a packet should be recirculated, its egress port has to be set to the recirculation port during processing in the ingress pipeline. The flow of a packet through the pipeline when it is recirculated is shown in Figure 7. The packet is still processed by the entire processing pipeline, i.e., the ingress pipeline and egress pipeline. However, after the packet has been deparsed, it is not sent through a regular physical egress port but pushed back into the switch-intern recirculation port. The packet is then processed as if it has been received on a physical port. The recirculation port has the same capacity

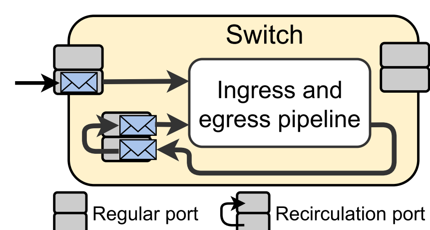


Figure 7. A packet is recirculated to a recirculation port and traverses the ingress and egress pipeline for a second time.

as the physical ports. For example, when two physical ports receive traffic at line rate and each packet is recirculated once, the recirculation port receives recirculated packets at double line rate, which causes packet loss.

2) RECIRCULATION CAPACITY

To discuss the effect of packet loss due to many recirculations we introduce the term 'recirculation capacity'. It is the available capacity to process recirculation traffic. Additional recirculation capacity is provided by using physical ports in loopback mode. When the forwarding device switches a packet to an egress port that is configured as a loopback port, the packet is immediately placed in the ingress of that port, instead. The packet is then processed as if it has been received on that port as usual, i.e., by the parser, the ingress and egress pipeline, and the deparser. Only traffic that has to be recirculated is switched to recirculation ports. In the following the term 'recirculation port' refers to a physical port in loopback mode, or the switch-intern recirculation port. When recirculation ports are required, the switch-intern recircution port should be used first, before any physical ports are configured as loopback ports. Only packets that are recirculated require recirculation capacity, i.e., common unicast traffic, e.g., as in regular IP unicast forwarding, is not recirculated, and therefore, does not occupy any recirculation capacity.

When multiple recirculation ports are deployed to increase the recirculation capacity, packets that should be recirculated need to be distributed over these ports. There are different distribution strategies. We developed a round-robin-based distribution approach for recirculation traffic to distribute the load equally over all recirculation ports. We store in a register which recirculation port receives the next packet which should be recirculated. When a packet has to be sent to a recirculation port, that register is accessed and updated in one atomic operation. This prevents any race conditions when traffic is distributed. Thus, this distribution strategy has two advantages. First, if n recirculation ports are used, the available recirculation capacity is increased to $n \cdot linerate$. Second, the equal distribution of recirculation traffic over all recirculation ports guarantees the full utilization of available recirculation capacities before packet loss occurs.

V. P4 IMPLEMENTATION OF BIER AND BIER-FRR FOR TOFINO

In this section we give an overview of the P4 implementation of BIER and tunnel-based BIER-FRR. First, we discuss the implementation basis. Afterwards, we give an overview of the processing of BIER packets, in particular we discuss packet recirculation.

A. CODEBASE

In [2] we presented a software-based prototype of a $P4_{16}$ implementation of BIER and tunnel-based BIER-FRR for the P4 software switch bmv2. We provided a very detailed description of the P4 programs including MATs with match fields and action parameters, control blocks, and applied operations. The prototype and the controller are publicly available on GitHub.[2]

In this paper we refrain from including a detailed technical description of the implementation for the Tofino. However, the source code[3] can be accessed by anyone on GitHub. In the following, we only explain important aspects of the hardware-based implementation to facilitate the understanding of the evaluation in Section VI and the model derivations in Section VII.

B. BIER PROCESSING

First, we describe the implementation of regular BIER forwarding on the Tofino. Afterwards, we explain operation of tunnel-based BIER-FRR.

1) BIER FORWARDING

Figure 8 shows how a BIER packet is processed once in the packet processing pipeline.

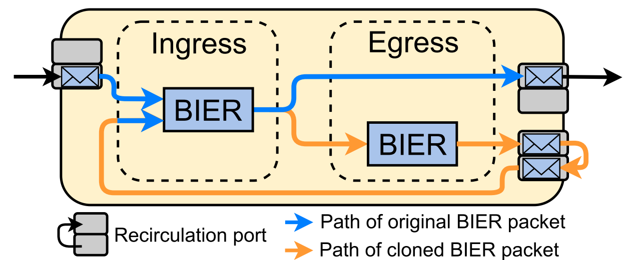


Figure 8. Paket flow of a BIER packet in the processing pipeline.

When the switch receives a BIER packet it is processed by the BIER control block. First, the BitString of the packet is matched against the BIFT which determines the egress port and the F-BM. The F-BM is applied to the BitString of the packet and cleared from the remaining bits. If the remaining bits still contain activated bits, CI2E is called and the egress port is set to a recirculation port so that the packet will be processed again. After the ingress pipeline, the copy is created and both packet instances enter the egress pipeline independently of each other. The original packet is sent through an egress port towards its NH. The packet clone is processed by a second BIER control block in the egress pipeline which sets the BitString of the packet copy to the remaining bits. Since the egress port of the packet clone is a recirculation port, the packet is recirculated, i.e., it is processed by the ingress pipeline again.

BIER forwarding removes BIER headers from packets that leave the BIER domain, and adds IP headers for tunneling through the routing underlay by tunnel-based BIER-FRR. Whenever a header is added or removed, the packet is recirculated for further processing.

When a BIER packet has more than one NH, two challenges appear. First, the BitString of a BIER packet has to be

[2]https://github.com/uni-tue-kn/p4-bier
[3]https://github.com/uni-tue-kn/p4-bier-tofino

matched several times against the BIFT to determine all NHs. However, matching a packet multiple times against the same MAT is not possible in P4. Second, multiple packet copies have to be created for forwarding. However, P4 does not allow to dynamically generate more than one copy of a packet. Therefore, we implemented a packet processing behavior where in each pipeline iteration one packet is forwarded to a NH and a copy of the packet is recirculated for further processing. This is repeated until all NHs receive a packet over which at least one destination of the BIER packet is reached. Figure 9 shows the processing of a BIER packet which has to be forwarded to three neighbors. In the first and second pipeline iteration the original BIER packet is sent through a physical egress port towards a NH and the copied BIER packet is recirculated by sending the packet copy to a recirculation port. In the last iteration when the remaining bits contain no activated bits anymore, no further packet copy is required and only the original BIER packet is sent through the egress port. In total, the packet needs to be recirculated two times to forward it to all three NHs. Therefore, in general, a BIER packet with n NHs, has to be recirculated $n-1$ times and the first NH can be served without packet recirculation.

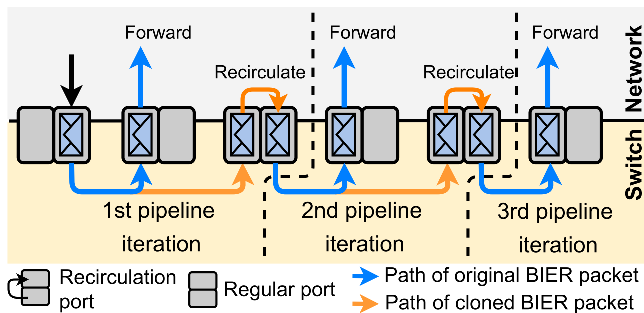


Figure 9. **BIER processing over multiple pipeline iterations.**

2) FORWARDING WITH TUNNEL-BASED BIER-FRR

The concept of tunnel-based BIER-FRR has been proposed in [2]. We implement it for the Tofino as follows.

The switch monitors the status of its ports as described in Section. When the match on the BIFT results in a NH which is reached by a port that is currently down, the processing of the BIER packet differs in the following way from the BIER processing described above. An IP header is added to the original BIER packet to tunnel the packet through the routing underlay towards an appropriate node in the BIER distribution tree. The egress port of the original packet is set to a recirculation port to process the IP header in another pipeline iteration, i.e., forward the IP packet to the right NH.

VI. PERFORMANCE EVALUATION OF THE P4-BASED HARDWARE PROTOTYPE

In this section we perform experiments to evaluate the performance of the P4-based hardware prototype for BIER regarding Layer-2 throughput and failover time, i.e., the time until BIER traffic is successfully delivered after a network failure.

A. FAILOVER TIME FOR BIER TRAFFIC

Here we evaluate the restoration time after a failure in three scenarios and vary the protection properties of IP and BIER. First, only the IP FIB and BIER FIB are updated by the controller, respectively, and no FRR mechanisms are activated. This process is triggered by a device that detects a failure. It notifies the controller which computes new forwarding rules and updates the IP and BIER FIB of affected devices. This scenario measures the time until the BIER FIB is updated after a failure, which is our baseline restoration time. The control plane, i.e., the controller, is directly connected to the P4 switch, which keeps the delay to a minimum in comparison to networks where the controller is several hops away.

Second, only BIER-FRR is deployed. In this scenario BIER is able to utilize tunnel-based BIER-FRR in case of a failure. However, FRR for IP traffic remains deactivated. Thus, IP traffic can be forwarded only after the IP FIB is updated.

Third, both IP-FRR and BIER-FRR are deployed. This scenario evaluates how quickly the P4 switch can react to network failures and restore connectivity of BIER and IP forwarding.

In the following, we first explain the setup and the metric. Then, we present our results. Finally, we discuss the influence of the setup on the results.

1) EXPERIMENT SETUP

Figure 10 shows the testbed. The Tofino [6], a P4-programmable switching ASIC, is at the core of the hardware testbed. We utilize a Tofino based Edgecore Wedge 100BF-32X [7] switch with 32 100 Gb/s ports. An EXFO FTB-1 Pro [42] 100 Gb/s traffic generator is connected to the Tofino to generate a data stream that is as precise as possible. Furthermore, we deploy two bmv2s that act as BFRs and BFERs. The traffic generator, the controller and two bmv2s are connected to the Tofino. The traffic generator sends IPMC traffic to the Tofino. The IPMC traffic has been subscribed only by bmv2-1. As long as the link between the Tofino and bmv2-1 works, the BIER packets are forwarded on the primary path. When the Tofino detects a failure, it notifies the

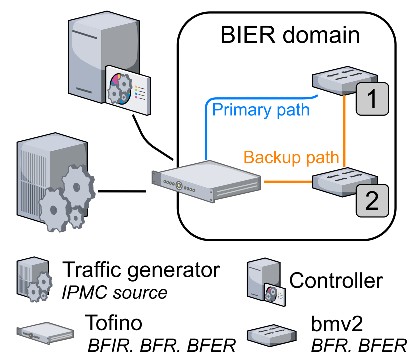


Figure 10. **Experimental setup for evaluation of restoration time.**

controller which computes new rules and updates forwarding entries of affected devices. In the meantime, the Tofino uses BIER-FRR to protect BIER traffic, and IP-FRR to protect IP traffic if enabled. This causes the Tofino to forward traffic on the backup path via bmv2-2 towards bmv2-1.

2) METRIC

We disable the link between the Tofino and bmv2-1 and measure the time until bmv2-1 receives BIER traffic again. We evaluate different combinations with and without IP-FRR and with and without BIER-FRR. To avoid congestion on the bmv2 and the VMs, the traffic generator sends only with 100 Mb/s, which has no impact on the results.

Figure 11 shows the average restoration time for the different deployed protection scenarios based on 10 runs which we discuss in the following. Confidence intervals are given on the base of a confidence level of 95%.

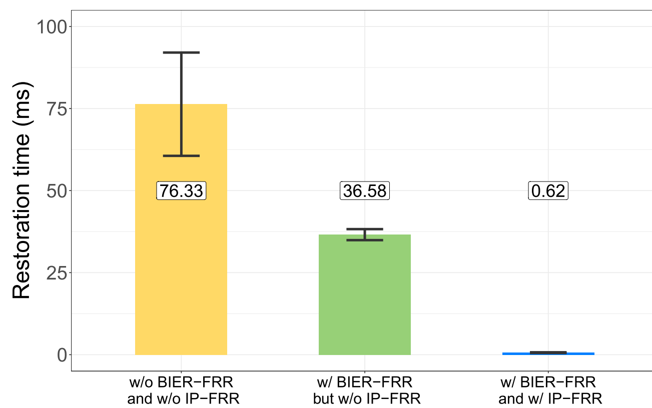


Figure 11. **Restoration time for BIER with different FRR strategies.**

3) FAILOVER TIME W/O BIER-FRR AND W/O IP-FRR

When no FRR mechanism is activated, multicast traffic arrives at the host only after the IP and BIER forwarding rules have been updated, which takes about 76 ms. The controller is directly connected to the Tofino. In a real deployment the controller may be multiple hops away, which would increase the restoration time significantly.

The same failover time is achieved without BIER-FRR but with IP-FRR, for which we do not present separate results. As BIER forwarding entries are updated only after IP forwarding entries have been updated, the use of IP-FRR in the network does not shorten the failover time for BIER traffic.

4) FAILOVER TIME W/BIER-FRR BUT W/O IP-FRR

When tunnel-based BIER-FRR but not IP-FRR is activated, bmv2-1 receives multicast traffic after 36 ms. In case of a failure, BIER-FRR tunnels the BIER traffic through the routing underlay. As soon as IP forwarding rules are updated, multicast traffic arrives at the host again. Since IP rules are updated faster than BIER rules, BIER-FRR decreases the restoration time for multicast traffic even if no IP-FRR mechanism is deployed.

5) FAILOVER TIME W/BIER-FRR AND W/IP-FRR

In the fastest and most resilient deployment both BIER-FRR and IP-FRR are activated. Then, multicast packets arrive at the host with virtually no delay after only 0.6 ms. In contrast to the previous scenario, unicast traffic is rerouted by IP-FRR which immediately restores connectivity for IP traffic.

6) INFLUENCE OF EXPERIMENTAL SETUP

The experimental setup (see Figure 10) features two BFERs on the base of bmv2 software switches with rather low performance compared to the Tofino-based hardware switch. However, we designed the experiment such that the low performance of these BFERs has no impact on results. bmv2 software switches can forward traffic with a rate up to 900 Mb/s [29]. By limiting the generated traffic rate to 100 Mb/s, the bmv2 switches forwarding and receiving BIER traffic are not overloaded so that bmv2-1 is able to measure correct restoration times. Furthermore, failure detection and protection switching are only carried out by the Tofino-based switch in the setup.

We now consider the impact of the hardware hosting the controller. When the controller is notified about a failure, it recomputes entries for IP and BIER forwarding tables. The computation time depends on the performance of the host and the size of the network in terms of number of nodes. Thus, the recomputation time may be significantly larger in larger networks, which increases the restoration time for BIER without any fast-reroute and for BIER with BIER-FRR but without IP-FRR. In contrast, the restoration time for BIER with BIER-FRR and IP-FRR is not impacted by the controller hardware or network size.

We discuss the impact of the signalling delay between the failure-detecting node and the controller. This delay was very low in our setup while it may be significantly larger in networks with large geographic extension or slow links. Such signalling delay adds to the restoration time for BIER without any fast-reroute and for BIER with BIER-FRR but without IP-FRR. The restoration time for BIER with BIER-FRR and IP-FRR is not impacted by that delay.

Finally, controller overload may occur when the controller needs to process too many messages, e.g., in case of a failure. This again has no impact on the restoration time for BIER with BIER-FRR and IP-FRR while it has significant impact on the restoration time for the other two settings.

B. THROUGHPUT FOR BIER TRAFFIC

The P4-based implementation of BIER described in Section V-B requires recirculation and is limited by the amount of recirculation capacity. The PSA defines a virtual port for this purpose. In this section we show the impact of insufficient recirculation capacity on throughput and the effect when additional physical recirculation ports, i.e., ports in loopback mode, are used for recirculation. We validate our experimental results in Section VI-C based on a theoretical model.

1) EXPERIMENTAL SETUP

The experimental setup is illustrated in Figure 12. A source node sends IPMC traffic to a BFIR. The BFIR encapsulates that traffic and sends it to a BFR. The BFR forwards the traffic to n BFERs which decapsulate the BIER traffic and send it as normal IPMC traffic to connected subscribers.

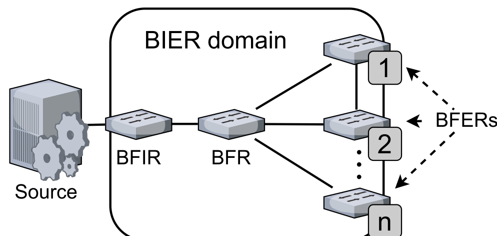


Figure 12. Theoretical setup for evaluation of BIER throughput.

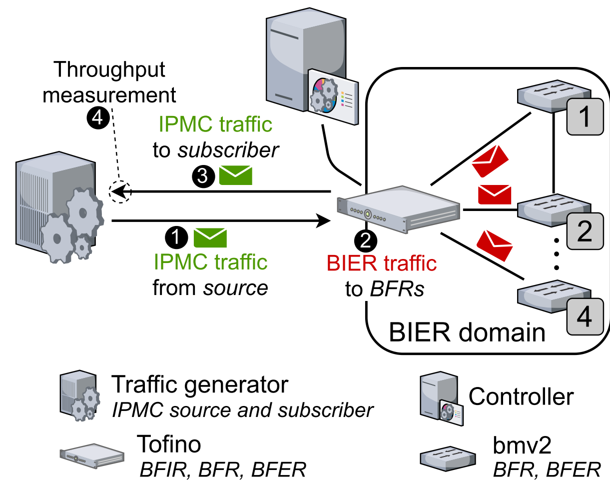


Figure 13. Hardware setup for evaluation of BIER throughput.

The goal of the experiment is to evaluate the forwarding performance of the BFR depending on the number of NHs. With n NHs, BIER packets have to be recirculated $n-1$ times, and internal packet loss occurs if recirculation capacity does not suffice. The objective of the experiment is to measure the BIER throughput depending on the number of recirculation ports for which only physical loopback ports are utilized in the experiment. However, the n subscribers may see different throughput. The first BFER does not see any packet loss while the last BFER sees most packet loss. Therefore, we measure the rate of IPMC traffic received on Layer 2 at the last subscriber.

2) HARDWARE SETUP AND CONFIGURATION

Due to hardware restrictions in our lab, we utilize one traffic generator, one P4-capable hardware switch, and one server running multiple P4 software switches to build the logical setup sketched above. The hardware setup is shown in Figure 13. The traffic generator is the source of IPMC traffic and sends traffic to the BFIR. The traffic generator is also the subscriber of BFER n and measures the throughput of received IPMC traffic on Layer 2. The hardware switch acts as BFIR, BFR, and BFER n while BFERs 1 to $n-1$ are deployed as P4 software switches on the server. In addition, we collapse the BFIR and the BFR in the hardware switch so that packet forwarding from the BFIR to the BFR is not needed. Therefore, the traffic generator is the last NH of the BIER packet when it is processed by the BFR.

Packet recirculation is required after (1) encapsulation to enable further BIER processing, (2) decapsulation to enable further IP forwarding, and (3) BIER packet replication to enable BIER forwarding to additional NHs. We set up the hardware switch so that all recirculation operations in connection with encapsulation and decapsulation are supported by two dedicated ports in loopback mode and spend another k ports in loopback mode to support packet recirculation after packet replication. This models the competition for recirculation ports on a mere BFR as in the theoretical model.

The P4 software switches are bmv2s that run alongside our controller on VMs on a server with an Intel Xeon Scalable Gold 6134 (8x 3.2 GHz) and 4 x 32 GB RAM. The P4 hardware switch is a Tofino [6] inside an Edgecore Wedge 100BF-32X [7] which is a 100 Gb/s P4-programmable switch with 32 ports. The traffic generator is an EXFO FTB-1 Pro [42] which generates up to 100 Gb/s. All devices are connected with QSFP28 cables which transmit up to 100 Gb/s.

3) INFLUENCE OF EXPERIMENTAL SETUP

The presented setup contains only a single Tofino-based switch which is partitioned and utilized as a single BFIR/BFR and a single BFER. All other BFERs in this setting are software switches that support only significantly lower bit rates (900 Mb/s [29]) than the Tofino-based switch (100 Gb/s). However, this has no impact on results because we measure the rate received by the single BFER implemented on the Tofino-based hardware. Furthermore, packet loss by the low-performance software switches does not reduce the generated traffic rate as this is configured as a constant rate on the generator.

4) BIER THROUGHPUT MEASUREMENTS DEPENDING ON RECIRCULATION PORTS

The traffic generator sends IPMC traffic at a rate of 100 Gb/s to the hardware switch, the hardware switch encapsulates the IPMC traffic, forwards BIER traffic iteratively n-1 times to bmv2s, recirculates the BIER packet to process the last activated header bit, decapsulates the traffic as BFER n, and returns it back to the traffic generator, which measures the received IPMC rate on Layer-2. We start measuring only after a 30 seconds initialization phase to avoid any influences from the startup phase. After 30 seconds, the traffic generator measures for 60 seconds the traffic arriving from the Tofino and reports the average Layer-2 throughput. We repeated experiments 10 times and computed confidence intervals with a confidence level of 95%. Their width was less than 0.5% of the measured average and, therefore, invisible.

Table 2. Model predictions $T(i)$ for BIER throughput and measured values $M(i)$ (Gb/s); the latter are the same values as presented in Figure 14.

Number of NHs:	1 NH		2 NHs		3 NHs		4 NHs	
Recirculation ports	$T(1)$	$M(1)$	$T(2)$	$M(2)$	$T(3)$	$M(3)$	$T(4)$	$M(4)$
1	100	99.32	100	99.32	38.2	43.3	15.74	19
2	100	99.32	100	99.32	100	99.32	53.14	50.5
3	100	99.32	100	99.32	100	99.32	100	99.32

Therefore, we omit them in future figures and tables for better readability.

In our experiments, we consider 1, 2, 3, and 4 NHs and utilize 1, 2, and 3 ports in loopback mode to support recirculation for BIER forwarding. The results are compiled in Figure 14.

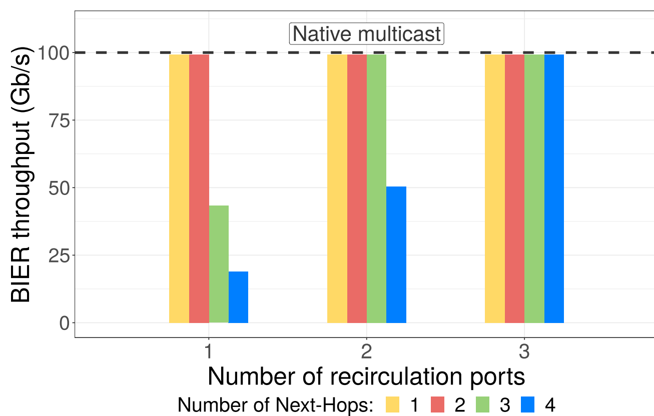


Figure 14. Measured throughput of BIER and traditional IPMC on the 100 Gb/s Tofino-based switch for different numbers of NHs and recirculation ports.

The left-most bar shows that with a single recirculation port, the last NH receives the full IPMC rate of 100 Gb/s if 1 NH is connected. The second bar from the left shows that the last NH still receives the full IPMC rate of 100 Gb/s if 2 NHs are connected. For 3 or 4 NHs, i.e., the third and fourth bar from the left, the IPMC traffic rate received by the last NH is reduced to 43 and 19 Gb/s, respectively.

With 2 recirculation ports, the last NH does not perceive a throughput degradation if at most 3 NHs, i.e., fifth to seventh bar from the left, are connected. For 4 NHs, i.e., eighth bar from the left, the IPMC traffic rate received by the last NH is reduced to 50 Gb/s.

And with 3 recirculation ports, even up to 4 NHs, i.e., ninth to twelfth bar from the left, can be supported without throughput degradation for the last NH.

Thus our experiments confirm that when multicast traffic arrives with 100 Gb/s at the Tofino, n-1 recirculation ports are needed to forward BIER traffic to n NHs without packet loss. This is different for a realistic multicast portion in the traffic mix, i.e., a minor fraction instead of 100%.

The hardware switch also supports traditional multicast in P4. With traditional multicast forwarding, all NHs receive 100 Gb/s regardless of the number of NHs. However, this comes with all the disadvantages of traditional IPMC we have discussed earlier.

C. THROUGHPUT MODEL FOR BIER FORWARDING WITH INSUFFICIENT RECIRCULATION CAPACITY

We model the throughput of BIER forwarding with insufficient recirculation capacity and validate the results with the experimentally measured values.

To forward a BIER packet to n NHs, it has to be recirculated $n-1$ times (see Section V-B). Any time a packet is sent to a recirculation, the packet is dropped with a certain probability if insufficient recirculation capacity is available. Due to the implemented round robin approach (see Section IV-C), the drop probability p is equal for all recirculation ports. The drop probability p in a system can be determined by comparing the available recirculation capacity and the sustainable recirculation load. The latter results from recirculations after BIER packet replication and takes packet loss into account. It is shown in the following formula.

$$C \cdot \sum_{m=1}^{n-1}(1-p)^m = k \cdot C \tag{1}$$

The available recirculation capacity is $k \cdot C$ where k is the number of recirculation ports and C is line capacity. The sustainable recirculation load is the sum of the successfully recirculated traffic rates after any number of recirculations. The traffic amount that has been successfully recirculated once is $C \cdot (1-p)$. The traffic amount that has been recirculated twice is $C \cdot (1-p)^2$, and so on. Therefore, the total amount is $C \cdot \sum_{m=1}^{n-1}(1-p)^m$.

We calculate the BIER throughput at any NH, i.e., after any number of recirculations. At the first NH, the throughput of the BIER traffic is C because the BIER packet is forwarded to the first NH before the packet is recirculated the first time. At the second NH, the BIER throughput is $C \cdot (1-p)$, at the third NH its $C \cdot (1-p)^2$, and so on. Therefore, the BIER throughput $T(i)$ at NH $1 \leq i \leq n$ is:

$$T(i) = C \cdot (1-p)^{i-1} \tag{2}$$

Table 2 shows the throughput predictions $T(i)$. We make predictions for the same scenarios as we evaluated in the performance evaluation in Section VI-B4 and compare them to the measured values $M(i)$.

The comparison shows that the model provides reasonable predictions for the BIER throughput.

VII. PROVISIONING RULE FOR RECIRCULATION PORTS

In this section we propose a provisioning rule for recirculation ports. It may be used for general P4-based applications requiring packet recirculation, not just for BIER forwarding. We first point out the importance for sufficient recirculation capacity. Then, we derive a general provisioning rule for recirculation ports and illustrate how their number depends on other factors. Finally, we apply that rule to provision the number of loopback ports for BFRs in the presence of traffic mixes.

A. IMPACT OF PACKET LOSS DUE TO MISSING RECIRCULATION CAPACITY

In Section IV-C we briefly discussed projects that leverage packet recirculation in P4. However, if recirculation capacity does not suffice and packets need to be recirculated several times, packet loss observed at the last stage may be quite high. We first illustrate this effect. If the packet loss probability due to missing recirculation capacity is p, then the overall packet loss probability after n recirculations is $p(n) = 1-(1-p)^n$. We illustrate this connection in Figure 15, which utilizes logarithmic scales to better view several orders of magnitude in packet loss. With only one recirculation, we obtain a diagonal for the overall packet loss. A fixed number of recirculations shifts the entire curve upwards, and with several recirculations like $n = 6$ or $n = 10$, the overall loss probability $p(6)$ or $p(10)$ is an order of magnitude larger than the packet loss probability p of a single recirculation step. Therefore, avoiding packet loss due to recirculations is important. Thus, sufficient recirculation capacity must be provisioned but overprovisioning is also costly since this means that entire ports at high speed cannot be utilized for operational traffic. Therefore, well-informed provisioning of recirculation ports is an important issue.

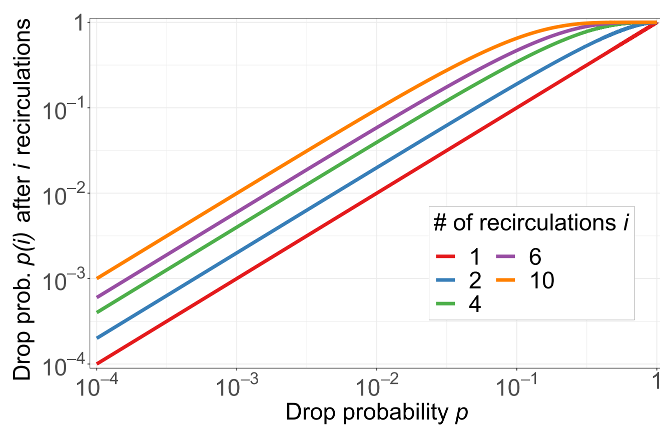


Figure 15. Loss probability after multiple recirculations.

B. DERIVATION OF A PROVISIONING RULE FOR RECIRCULATION PORTS

We first introduce the recirculation factor R and the utilization ratio U. Then, we use them to derive a provisioning rule for recirculation ports.

The recirculation factor R is the average number of recirculations per packet. Not all packets may be recirculated or the number how often a packet is recirculated depends on the particular packet.

The utilization ratio U describes the multiple by which a recirculation port can be higher utilized than a normal port. For example, if the average utilization of each normal port is 10%, then each recirculation port may be operated with a utilization of 40%, in particular if multiple of them are utilized. This corresponds to a utilization ratio of $U = 4$. We give some rationales for that idea. Normal ports at high speed are often underutilized in practice because bandwidths exist only in fixed granularities and usually link speeds are heavily overprovisioned to avoid upgrades in the near future. Furthermore, some links operate at lower utilization, others at higher utilization. Recirculation ports can be utilized to a higher degree. First, there is no need to keep the utilization of recirculation ports low for reasons like missing appropriate lower link speeds as it can be the case for normal ports. Second, recirculation ports are shared for all recirculation traffic of a switch so that resulting traffic fluctuations are lower and the utilization of the ports can be higher than the one of other ports.

If m incoming ports carry traffic with a recirculation factor R and a utilization ratio U can be used on the switch, then

$$m' = \left\lceil \frac{m \cdot R}{U} \right\rceil \tag{3}$$

describes the number of required recirculation ports.

C. ILLUSTRATION OF REQUIRED RECIRCULATION PORTS

For illustration purposes, we consider a P4 switch with 32 physical (external) ports and one virtual (internal) port in loopback mode for recirculations. If the capacity of that single virtual recirculation port does not suffice for recirculations, physical ports need to be turned into loopback mode as well and be used for recirculation. All recirculation ports are utilized in round-robin manner to ensure equal utilization among them.

Thus, the number of normal ports m plus the number of recirculation ports m' must be at most 33, i.e., 32 physical ports and 1 virtual port. Therefore, we find the smallest m' according to Equation 3, so that $m+m' \leq 33$ while maximizing m. The number of physical recirculation ports is $m'-1$ as the virtual port can also be used for recirculations. Figure 16 shows the number of physical recirculation ports depending on the recirculation factor R and the utilization ratio U. Since U depends on the specific use case and traffic mix, we present results for different values of U. Thereby, R and U are fractional numbers. While the number of recirculations for each packet is an integral number, the average number of recirculations per packet R is fractional. The number of physical recirculation ports increases with the recirculation factor R. Due to the fact that both m and m' are integers, the number of physical recirculation ports $(m - 1)$ is not monotonously increasing because for some R and U the sum

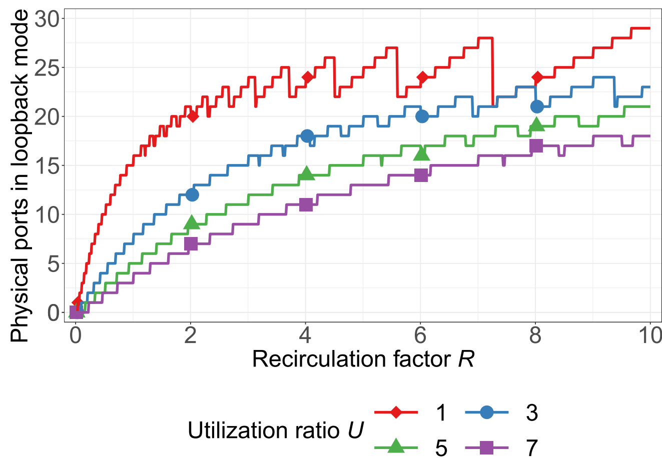


Figure 16. Number of physical ports in loopback mode.

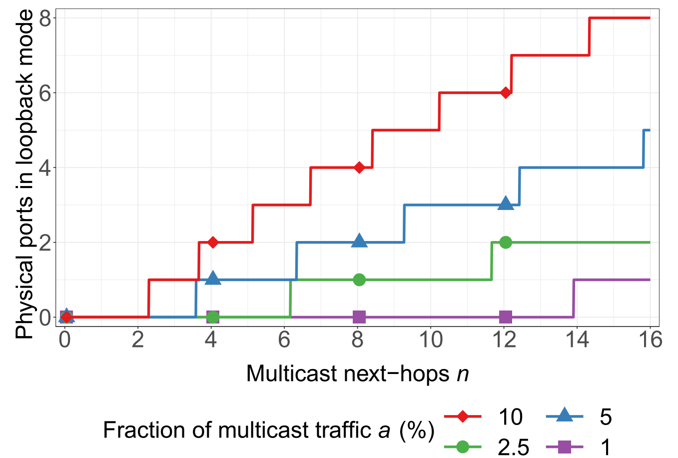


Figure 17. Physical ports in loopback mode for traffic mixes with realistic multicast portions.

$m + m'$ amounts to the maximum 33, and to lower values for other R and U.

The various curves show that the number of required physical recirculation ports decreases with increasing utilization ratio U. With a large recirculation factor $R \geq 3$ and a low utilization Ratio $U \leq 3$, half of the ports of the 32 port switch or even more need to be used for recirculation, which is expensive. However, with small $R < 1$ and large $U > 3$ the number of required physical recirculation ports is low because most of the traffic does not require packet recirculation, and due to the large utilization ratio U, the recirculation ports can cover significantly more traffic than normal ports. It is even possible that no physical recirculation port is needed if the recirculation capacity of the internal recirculation port can cover the recirculation load.

D. APPLICATION OF THE PROVISIONING METHOD TO TRAFFIC MIXES WITH BIER

In this section we make predictions for m', the number of recirculation ports, for traffic mixes with typical multicast portions. We assume different portions of multicast traffic $a \in \{0.01, 0.025, 0.05, 0.1\}$ and different average numbers of BIER NHs $n \in \{0, 2, 4, \ldots, 16\}$, i.e., each BIER packet is recirculated $n - 1$ times on average. Since unicast traffic is normally processed without recirculation, it does not need any recirculation capacity, i.e., its amount has no influence on the number of required recirculation ports and is, therefore, not considered in this analysis. Then, we calculate $R = a \cdot (n - 1)$, and assume $U = 4$. Again, we calculate the smallest m', i.e., like in Equation 3, so that $m + m' \leq 33$ while maximizing m. Figure 17 shows the number of physical recirculation ports depending on the average number of multicast NHs n and the fraction of multicast traffic a. If the fraction of multicast traffic is low like 1%, the capacity of the internal port suffices to serve up to 13 NHs on average. Moderate fractions of 2.5% multicast traffic require no physical recirculation port for up to 5 NHs, 1 physical recirculation port for up to 11 NHs, and 2 physical recirculation ports for 12 and more NHs. With 5% multicast traffic, the number of required physical recirculation ports increases almost linearly from zero to 5 with an increasing number of NHs. Large fractions of multicast traffic, like 10%, require up to 8 recirculation ports if the number of NHs is also large like 16. Under such conditions, 25% of the physical ports cannot be used for normal traffic forwarding as they are turned into loopback mode. However, the assumptions seem rather unlikely as multicast traffic typically makes up only a small proportion of the traffic.

VIII. CONCLUSION

The scalability of traditional IPMC is limited because core devices need to maintain IPMC group-dependent forwarding state and process lots of control traffic whenever topology or subscriptions change. Therefore, BIER has been introduced by the IETF as an efficient transport mechanism for IPMC traffic. State in BIER core devices does not depend on IPMC groups, and control traffic is only sent to border nodes, which increases scalability in comparison to traditional IPMC significantly. In addition, there are fast-reroute (FRR) mechanisms for BIER to minimize the effect of network failures. However, BIER cannot be configured on legacy devices as it implements a new protocol with a complex forwarding behavior.

In this paper we demonstrated a P4-based implementation of BIER with tunnel-based BIER-FRR, IP unicast with FRR, IP multicast, and Ethernet forwarding. The target platform is the P4-programmable switching ASIC Tofino which is used in the Edgecore Wedge 100BF-32X, a 32 100 Gb/s port high-performance P4 switch.

In an experimental study, we showed that BIER-FRR significantly reduces the restoration time after a failure, and in combination with IP-FRR, the restoration time is reduced to less than 1 ms. We confirmed that the prototype is able to forward traffic at a speed up to 100 Gb/s. However,

under some conditions, less throughput is achieved when switch-internal recirculation ports are overloaded. As a remedy, we added more recirculation capacity by turning physical ports into loopback mode. We modelled BIER forwarding, predicted limited throughput due to missing recirculation capacity, and validated the results by measured values. Furthermore, we proposed a simple method for provisioning of physical recirculation ports. The approach was motivated by BIER, but holds for general P4 programs requiring recirculations. In a case study, we applied it to BIER with different mixes of unicast and multicast traffic and showed that only a few physical recirculation ports suffice under realistic conditions.

REFERENCES

[1] I. Wijnands. (Nov. 2017). *RFC 8279: Multicast Using Bit Index Explicit Replication (BIER)*. [Online]. Available: https://datatracker.ietf.org/doc/rfc8279/

[2] D. Merling, S. Lindner, and M. Menth, "P4-based implementation of BIER and BIER-FRR for scalable and resilient multicast," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102764.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, and J. Rexford, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[4] W. Braun, J. Hartmann, and M. Menth, "Scalable and reliable software-defined multicast with BIER and P4," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 905–906.

[5] P4lang. (2021). *behavioral-Model*. Accessed: Jan. 28, 2021. [Online]. Available: https://github.com/p4lang/behavioral-model

[6] Edge-Core Networks. (2017). *The World's Fastest & Most Programmable Networks*. [Online]. Available: https://barefootnetworks.com/resources/worlds-fastest-most-programmable-networks/

[7] Edge-Core Networks. (2019). *Wedge100BF-32X/65X Switch*. [Online]. Available: https://www.edge-core.com/_upload/images/Wedge100BF-32X_65X_DS_R05_2019%1210.pdf

[8] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source routed multicast for public clouds," in *Proc. ACM Special Interest Group Data Commun.*, 2019, pp. 458–471.

[9] S. Islam, N. Muslim, and J. W. Atwood, "A survey on multicasting in software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 355–387, 1st Quart., 2018.

[10] Z. AlSaeed, I. Ahmad, and I. Hussain, "Multicasting in software defined networks: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 104, pp. 61–77, Feb. 2018.

[11] J. Ráckert, J. Blendin, and D. Hausheer, "Software-defined multicast for over-the-top and overlay-based live streaming in ISP networks," *J. Netw. Syst. Manage.*, vol. 23, no. 2, pp. 280–308, Apr. 2015.

[12] J. Ruckert, J. Blendin, R. Hark, and D. Hausheer, "Flexible, efficient, and scalable software-defined over-the-top multicast for ISP environments with DynSdm," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 4, pp. 754–767, Dec. 2016.

[13] T. Humernbrum, B. Hagedorn, and S. Gorlatch, "Towards efficient multicast communication in software-defined networks," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Jun. 2016, pp. 106–113.

[14] C. A. S. Oliveira, "Steiner trees and multicast," *Math. Aspects Netw. Routing Optim.*, vol. 53, pp. 29–45, Dec. 2011.

[15] L.-H. Huang, H.-J. Hung, C.-C. Lin, and D.-N. Yang, "Scalable and bandwidth-efficient multicast for software-defined networks," in *Proc. IEEE Global Commun. Conf.*, Dec. 2014, pp. 1890–1896.

[16] Z. Hu, D. Guo, J. Xie, and B. Ren, "Multicast routing with uncertain sources in software-defined network," in *Proc. IEEE/ACM 24th Int. Symp. Qual. Service (IWQoS)*, Jun. 2016, pp. 1–6.

[17] S. Zhou, H. Wang, S. Yi, and F. Zhu, "Cost-efficient and scalable multicast tree in software defined networking," in *Proc. Conf. Algorithms Archit. Parallel Process.*, 2015, pp. 562–605.

[18] J.-R. Jiang and S.-Y. Chen, "Constructing multiple Steiner trees for software-defined networking multicast," in *Proc. 11th Int. Conf. Future Internet Technol.*, Jun. 2016, pp. 1–6.

[19] B. Ren, D. Guo, J. Xie, W. Li, B. Yuan, and Y. Liu, "The packing problem of uncertain multicasts," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 16, p. e3985, Aug. 2017.

[20] Y.-D. Lin, Y.-C. Lai, H.-Y. Teng, C.-C. Liao, and Y.-C. Kao, "Scalable multicasting with multiple shared trees in software defined networking," *J. Netw. Comput. Appl.*, vol. 78, pp. 125–133, Jan. 2017.

[21] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data center multicast using software defined networking," in *Proc. 6th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2014, pp. 1–8.

[22] W. Cui and C. Qian, "Scalable and load-balanced data center multicast," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 1–6.

[23] W.-K. Jia and L.-C. Wang, "A unified unicast and multicast routing and forwarding algorithm for software-defined datacenter networks," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2646–2657, Dec. 2013.

[24] M. J. Reed, M. Al-Naday, N. Thomos, D. Trossen, and G. Petropoulos, "Stateless multicast switching in software defined networks," in *Proc. IEEE Int. Conf. Commun.*, May 2016, pp. 1–7.

[25] S.-H. Shen, L.-H. Huang, D.-N. Yang, and W.-T. Chen, "Reliable multicast routing for software-defined networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 181–189.

[26] M. Popovic, R. Khalili, and J.-Y. Le Boudec, "Performance comparison of node-redundant multicast distribution trees in SDN networks," in *Proc. Int. Conf. Networked Syst. (NetSys)*, Mar. 2017, pp. 1–8.

[27] D. Kotani, K. Suzuki, and H. Shimonishi, "A multicast tree management method supporting fast failure recovery and dynamic group membership changes in OpenFlow networks," *J. Inf. Process.*, vol. 24, no. 2, pp. 395–406, 2016.

[28] T. Pfeiffenberger, J. L. Du, P. B. Arruda, and A. Anzaloni, "Reliable and flexible communications for power systems: Fault-tolerant multicast with SDN/OpenFlow," in *Proc. 7th Int. Conf. New Technol., Mobility Secur. (NTMS)*, Jul. 2015, pp. 1–6.

[29] A. Bas. (Jan. 2018). *BMv2 Throughput*. [Online]. Available: https://github.com/p4lang/behavioral-model/issues/537#issuecomment-360537441

[30] A. Giorgetti, A. Sgambelluri, F. Paolucci, P. Castoldi, and F. Cugini, "First demonstration of SDN-based bit index explicit replication (BIER) multicasting," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2017, pp. 1–6.

[31] A. Giorgetti, A. Sgambelluri, F. Paolucci, N. Sambo, P. Castoldi, and F. Cugini, "Bit index explicit replication (BIER) multicasting in transport networks," in *Proc. Int. Conf. Opt. Netw. Design Modeling (ONDM)*, May 2017, pp. 1–5.

[32] Y. Desmouceaux and T. Clausen, "Reliable multicast with BIER," *J. Commun. Netw.*, vol. 20, pp. 182–197, May 2018.

[33] T. Eckert. (Nov. 2017). *Traffic Engineering for Bit Index Explicit Replication BIER-TE*. [Online]. Available: http://tools.ietf.org/html/draft-eckert-bier-te-arch

[34] W. Braun, M. Albert, T. Eckert, and M. Menth, "Performance comparison of resilience mechanisms for stateless multicast using BIER," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 230–238.

[35] F. Hauser, M. Haeberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: Fundamentals, advances, and applied research," 2021, *arXiv:2101.10632*. [Online]. Available: https://arxiv.org/abs/2101.10632

[36] J. Geng, J. Yan, and Y. Zhang, "P4QCN: Congestion control using P4-capable device in data center networks," *Electronics*, vol. 8, p. 280, Mar. 2019.

[37] C. Wernecke, H. Parzyjegla, G. Muhl, P. Danielis, and D. Timmermann, "Realizing content-based publish/subscribe with P4," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2018, pp. 1–7.

[38] M. Uddin, S. Mukherjee, H. Chang, and T. V. Lakshman, "SDN-based multi-protocol edge switching for IoT service automation," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2775–2786, Dec. 2018.

[39] *Reprinted from Journal of Network and Computer Applications, vol. 169, Daniel Merling, Steffen Lindner, Michael Menth, P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast*, Elsevier, Amsterdam, The Netherlands, 2020.

[40] D. Katz. (Jul. 2004). *Bidirectional Forwarding Detection (BFD)*. [Online]. Available: https://datatracker.ietf.org/doc/rfc5880/

[41] Q. Xiong. (Oct. 2017). *BIER BFD*. [Online]. Available: https://datatracker.ietf.org/doc/draft-hu-bier-bfd/

[42] EXFO. (2019). *FTB-1v2/FTB-1 Pro Platform*. [Online]. Available: https://www.exfo.com/umbraco/surface/file/download/?ni=10900&cn=en-US&pi=5404

DANIEL MERLING received the master's degree from the Chair of Communication Networks of Prof. Dr. habil. Michael Menth, Eberhard Karls University, Tübingen, Germany, in 2017, where he is currently pursuing the Ph.D. degree. His research interests include software-defined networking, scalability, P4, routing and resilience issues, multicast, and congestion management.

STEFFEN LINDNER received the bachelor's and master's degrees from the Chair of Communication Networks of Prof. Dr. habil. Michael Menth. He is currently pursuing the Ph.D. degree with the Communication Networks Research Group, Eberhard Karls University, Tübingen, Germany. His research interests include software-defined networking, P4, and congestion management.

MICHAEL MENTH (Senior Member, IEEE) received the Diploma degree from The University of Texas at Austin, in 1998, the Ph.D. degree from the University of Ulm, Germany, in 2004, and the Habilitation degree from the University of Wuerzburg, Germany, in 2010. He is currently a Professor with the Department of Computer Science, University of Tuebingen, Germany, since 2010, and the Chair Holder of communication networks. His research interests include performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, industrial networking and the Internet of Things, software-defined networking, and the Internet protocols.

1.7 Efficiency of BIER Multicast in Large Networks

Efficiency of BIER Multicast in Large Networks

Daniel Merling*, Thomas Stüber*, Michael Menth

Chair of Communication Networks, University of Tuebingen, Germany
{daniel.merling, thomas.stueber, menth}@uni-tuebingen.de

***Abstract*—Bit Index Explicit Replication (BIER) has been introduced by the IETF to transport IP multicast (IPMC) traffic within a BIER domain. Its advantage over IPMC is improved scalability regarding the number of multicast groups. However, scaling BIER to large networks is a challenge. To that end, receivers of a BIER domain are assigned to smaller subdomains. To deliver an IPMC packet over a BIER domain, a copy is sent to any subdomain with a receiver for that packet. Consequently, some links may carry multiple copies of the same IPMC packet, which contradicts the multicast idea.**

In this paper, we propose and compare various algorithms to select subdomains for BIER in order to keep the overall BIER traffic low despite multiple packet copies. We apply them to investigate the traffic savings potential of IPMC and BIER relative to unicast under various conditions. We show that the traffic savings depend on network topology, network size, and the size of the multicast groups. Also the extra traffic caused by BIER depends on these factors. In spite of some redundant packets, BIER can efficiently reduce the overall traffic in most network topologies. Similarly to IPMC, BIER also avoids heavily loaded links. Finally, we demonstrate that BIER subdomains optimized for failure-free conditions do not cause extensive overload in case of single link failures.

***Index Terms*—Bit Index Explicit Replication (BIER), multicast, IP networks, performance evaluation, optimization**

I. Introduction

IP multicast (IPMC) distributes traffic of a multicast group along a tree so that any link in an IP network forwards at most a single copy of a packet. However, each multicast group requires signalling and forwarding state in all routers of the tree. In case of subscriber change, updates to the forwarding states of a single multicast group may be required. In case of link or node failures, the traffic of many multicast groups may be affected so that routers experience a large signalling load. The IETF has proposed Bit Index Explicit Replication (BIER) [1] to counteract that problem. BIER tunnels multicast traffic through a BIER domain and delivers a copy to each desired egress node. BIER solves the scalability problem by keeping the core nodes of the BIER domain unaware of any multicast group. Nevertheless, scaling BIER to large networks is a challenge. Multiple copies of a multicast packet may need to be forwarded over the same link, which contracts the multicast idea and may prevent BIER from efficiently reducing the traffic load for multicast traffic. We briefly explain the reason and provide the ground for this research work.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

*These authors contributed equally

When an ingress node of a BIER domain receives an IPMC packet, it adds a BIER header including a bitstring. The positions in the bitstring correspond to egress nodes of the BIER domain and the activated bits indicate the receivers of the BIER packet. The bitstring enables BIER routers to forward BIER packets without knowing individual multicast groups. As the bitstring has a limited size, BIER domains with more egress nodes require a scaling feature. Subdomains are introduced which are sets of egress nodes, and bitstrings are defined for each subdomain. Thus, if an IPMC packet needs to be forwarded to egress nodes in different subdomains, multiple BIER packets with different bitstrings are sent and possibly pass identical links. This obviously diminishes the efficiency of BIER to reduce the load of multicast traffic compared to IPMC.

In this paper we quantify and compare the ability of IPMC and BIER to efficiently reduce the load from multicast traffic in comparison to unicast. We define suitable metrics and show that the efficiency of IPMC depends on network topology and size as well as the size of the multicast groups. As the BIER standard [1] does not specify the computation of subdomains, we propose various algorithms for that purpose. Thereby, the objective is to minimize the resulting load of BIER traffic. We utilize these algorithms to define suitable subdomains for BIER networks and use them to assess the ability of BIER to efficiently transport multicast traffic. As the subdomains are optimized for failure-free conditions, we also investigate their effect on traffic distribution in case of link failures.

The remainder of the paper is structured as follows. In the next section we review related work. Section III gives a primer on BIER and shows that BIER generates a separate packet copy for almost every subdomain even for small multicast groups. In Section IV we propose algorithms to compute subdomains for BIER networks. We compare the algorithms with regard to runtime and quality in Section V. Section VI evaluates and compares the traffic savings potential of IPMC and BIER for multicast traffic under various conditions. In Section VII we evaluate the efficiency of BIER in case of single link failures. Finally, we conclude the paper in Section VIII.

II. Related Work

We review advances for IPMC and BIER-based multicast and mention well-known clustering algorithms.

A. Advances for IPMC

Islam et al. [2] and Al-Saeed et al. [3] provide comprehensive surveys for multicast. Most of the cited papers discuss shortcomings of IPMC as already mentioned in Section I. Many approaches aim to enhance IPMC. Intelligent mechanisms for multicast tree-building are presented to reduce the size of the forwarding information base (FIB), or efficient signalling mechanisms are proposed.

Elmo [4] improves the scalability of traditional IPMC in data centers. Multicast group information is encoded in packet headers to reduce the FIB of core nodes by leveraging characteristic properties of data center topologies. The Avalanche Routing Algorithm (AvRA) [5] also leverages properties of data center networks to optimize link utilization of distribution trees. Dual-Structure Multicast (DuSM) [6] builds specialized forwarding structures for high-bandwidth and low-bandwidth flows. It improves scalability and link utilization in data centers.

Zhang et al. [7] optimize application layer multicast (ALM). They continuously monitor the application-specific distribution tree and update its structure according to the optimization objective of the multicast group. The authors of [8] study the distribution of delay-sensitive data with minimum latency. They propose a set of algorithms that construct minimum-delay trees for different kinds of application requirements like min-average, min-maximum, real-time requirements, etc. Li et al. [9] leverage the structure of data center networks to improve the scalability of traditional multicast. They optimize the forwarding tables by partitioning the multicast address space and aggregating multicast addresses at bottleneck switches. Kaafar et al. [10] present a new overlay multicast tree construction scheme. It leverages location-information of subscribers to build efficient distribution trees.

Software-Defined Multicast (SDM) [11] is a well-managed multicast platform. It is specialized on P2P-based video streaming for over-the-top and overlay-based live streaming services. In [12] traffic engineering features are added to SDM. Lin et al. [13] propose to share distribution trees between multicast groups to reduce the size of the FIB in core nodes and implement it in OpenFlow. Similarly, the authors of [14] leverage bloom filters to reduce the number of TCAM-entries in software-defined networks. Adaptive SDN-based SVC multicast (ASCast) [15] optimizes multicast forwarding for video live streaming by minimizing latency and delay. To that end, the authors propose an integer linear program for optimal tree building, and TCAM-based forwarding tables for fast packet processing. Humernbrum et al. [16] reduce the size of the FIB in some core nodes by introducing address translation from multicast addresses to unicast addresses at the last multicast hop. Jia et al. [17] reduce the size of the FIB in core nodes and facilitate efficient implementations. They leverage prime numbers and the Chinese remainder theorem to efficiently organize FIB structures. Steiner trees [18] are well-researched structures to build efficient multicast trees. Many papers modify and extend Steiner trees to build specialized multicast trees that minimize specific aspects like link costs [19], number of branch nodes [20], number of hops [21], delay [22], optimal placement of IPMC sources [23], or retransmission efficiency [24].

B. Advances for BIER

BIER uses a novel header and its forwarding behaviour distinguishes substantially from IP forwarding. Giorgetti et al. [25], [26] show a first implementation of BIER in OpenFlow. Merling et al. [27] present a BIER prototype for a P4-programmable software switch with a throughput of around 900 Mb/s. In a follow-up work [28] they implement BIER for the P4-programmable switching ASIC Tofino that supports 100 Gb/s throughput per port. They also propose how BIER traffic should be rerouted in case of failures, which has been adopted as IETF working group document [29].

The authors of [30] evaluate the retransmission efficiency of BIER when subscribers signal missing packets by negative acknowledgements, i.e., NACKs. Traditional IPMC leverages either unicast packets or retransmission to the entire multicast group when some subscribers signal NACKs. The BIER header allows to retransmit packets to specific subscribers only, i.e., NACK senders, while sending only one packet copy over each link. The authors find that BIER causes less overhead in terms of number of retransmitted packets and that it achieves better link utilization. Desmouceaux et al. [31] increase efficiency of retransmission with BIER by allowing intermediate nodes to resend packets, if possible, instead of resending the packet at the source. This significantly reduces the overall retransmission traffic.

Eckert et al. [32] propose tree engineering for BIER, i.e., BIER-TE. It leverages the BIER header to also encode the distribution tree of a packet in terms of traversed links. In [33] 1+1 protection for BIER is presented using maximally redundant trees (MRTs). Traffic is distributed simultaneously over two disjoint trees so that packets are delivered even if one tree is compromised by a failure.

C. Clustering Algorithms

In this work we cluster receivers of BIER domains into subdomains. Karypis et. al. [34] present an algorithm to compute a bisection of a graph by performing a breadth-first search starting from two center nodes. The authors of [35] propose a similar method to compute k-partitions for arbitrary k, using k center nodes. Instead of two breadth-first searches, this algorithm performs k breadth-first searches in parallel. The resulting partitions tend to reduce the number of border nodes instead of cross-edges, which is a good property for load balancing. The approach is closely related to k-means clustering with Lloyd's algorithm [36]. The algorithm selects k center nodes and adds all nodes to the cluster with the nearest center node. The center nodes are readjusted to reflect the center of the clusters and this step is repeated until no changes occur. k-means clustering is not suitable for our problem, as cluster sizes cannot be limited. In contrast to that, the bubble growing approach of [35] produces equal size partitions. The

heuristic algorithm for BIER clustering in this work follows a similar approach.

III. Bit Index Explicit Replication (BIER)

In this section we introduce fundamentals of BIER and explain its scaling mechanism for large networks. In addition, we show that the mechanism tends to produce multiple BIER packets for a single IPMC packet, even for small multicast groups.

A. Overview

BIER is a domain-based mechanism to transport IPMC traffic over a so-called routing underlay network, e.g., an IP network [1]. Figure 1 shows the layered BIER architecture.

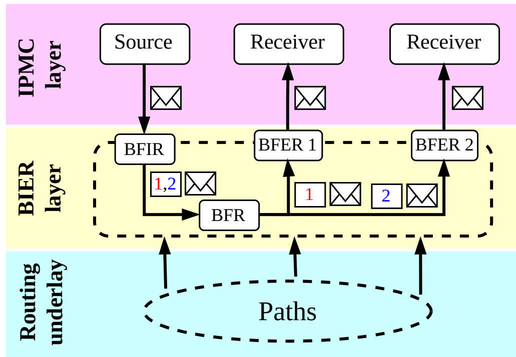


Fig. 1: Layered BIER architecture according to [27].

BIER-capable routers are called bit forwarding routers (BFRs). Ingress and egress nodes of a BIER domain are called bit forwarding ingress and egress routers (BFIRs, BFERs). The BIER header contains a bitstring with bit positions for all BFERs. BFIRs encapsulate IPMC traffic with a BIER header and the activated bits in its bitstring indicate the set of BFERs that should receive a copy of the packet. BFRs forward BIER packets based on this bitstring along a tree towards the indicated BFERs. Thereby, only a single copy is sent over each involved link. The paths of the tree are inherited from the routing underlay but BIER-encapsulated IPMC packets are usually sent over Layer 2 technology. BFERs remove the BIER header from the packets and pass them to the IPMC layer.

B. Scaling BIER to Large Networks

BIER hardware must implement a bitstring length of 256 bits, but larger bitstrings, e.g., 1024 bits, may also be supported [1]. However, large bitstrings increase the header size, which is tolerable only to some extent. Any BFER requires a position in the bitstring to be addressable. To make BIER applicable to networks with more BFERs than the size of the bitstring, so-called BIER subdomains are introduced that are identified in the BIER header by their subdomain identifier (SDI). Subdomains define different mappings of BFERs to bit positions for the bitstring so that only the combination of SDI and bitstring determines the addressed BFERs. If a BFIR receives an IPMC packet, it must send an encapsulated copy possibly for multiple subdomains to ensure that all desired BFERs are reached.

C. BIER Packets Needed for Single IPMC Packet

When a BIER domain is large, it may require multiple subdomains. Then, the BFERs of a BIER domain are assigned to bit positions in the bitstrings of different subdomains. As a consequence, when an IPMC packet is to be carried through a BIER domain, multiple BIER packets with different SDIs may be created to address all desired receivers. We call them redundant packet copies as they carry the same IPMC packet. They cause extra traffic and reduce BIER's ability to reduce load from multicast traffic compared to normal IPMC forwarding.

We investigate how many different BIER packets are generated on average when a BFIR sends an IPMC packet over a BIER domain. To that end, we consider a BIER domain with $n = 1024$ BFERs and bitstring lengths of $b \in \{128, 256, 512, 1024\}$ bits. Hence, $s \in \{1, 2, 4, 8\}$ subdomains are needed to provide all BFERs with bit positions. We use a Markov chain model to compute the average number of different BIER packets needed if an IPMC packet has r BFERs as receivers; thereby we assume that receivers of a packet belong with equal probability to any of the subdomains.

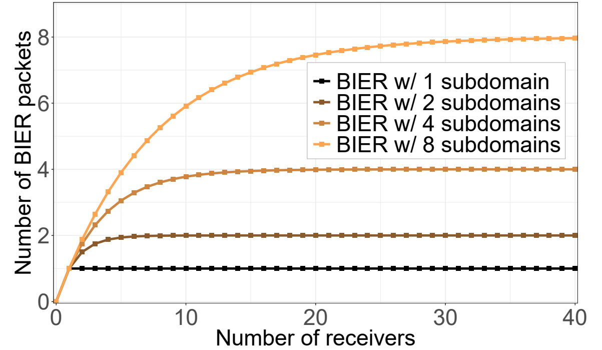


Fig. 2: Average number of redundant BIER packet copies needed to forward a single IPMC packet through a BIER domain with $n = 1024$ BFERs partitioned into $s \in \{1, 2, 4, 8\}$ subdomains.

Figure 2 shows that the average number of BIER packets significantly depends on the number of receivers r and the number of subdomains s. The number of BIER packets converges quickly to the number of subdomains s. If $r = 3 \cdot s$ receivers are addressed, almost s different BIER packets need to be sent for a single IPMC packet.

As the number of redundant packets is large even for small multicast groups, it is relevant to study their impact on the overall extra traffic in the network and on the ability of BIER to efficiently carry multicast traffic compared to IPMC. Moreover, the effect of redundant BIER packets may be mitigated. If a specific part of the BIER domain accommodates only BFERs from a single subdomain, only packets for that

subdomain will be forwarded to that part of the network, which avoids redundant packets in this area. Thus, when subdomains are chosen appropriately, BIER may be able to deliver multicast traffic with only little extra traffic compared to IPMC.

IV. Algorithms for BIER Clustering

As explained in the previous section, sudomains for BIER domains should be defined such that the overall load from multicast traffic is low in the entire network even if multiple redundant BIER packets need to be sent to BFERs in different subdomains.

We first formalize this challenge as the "BIER clustering problem". Then, we propose three classes of algorithms to assign BFERs to subdomains of a BIER domain: random, optimal, and heuristic. For optimal solutions, we propose topology-specific algorithms for selected, regular topologies. For arbitrary topologies we propose an integer linear program (ILP) to optimally solve the BIER clustering problem. Finally, we suggest a heuristic algorithm that may be used when the ILP is not solvable for complexity reasons.

A. The BIER Clustering Problem

We introduce nomenclature and constraints as well as the objective function for BIER clustering, and discuss alternate optimization goals.

1) Nomenclature and Constraints: A network topology is given by a set of n vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. The set of edges on the path between any two nodes $v, w \in \mathcal{V}$ is denoted by $p(v, w) \subseteq \mathcal{E}$; it is inherited by the routing underlay. The objective of the clustering is to find a set of subdomains $\mathcal{C}$ so that any subdomain $\mathcal{S} \in \mathcal{C}$ is a subset of all nodes $\mathcal{S} \subseteq \mathcal{V}$ and the union of all subsets covers all nodes, i.e., $\bigcup_{\mathcal{S}\in\mathcal{C}} \mathcal{S} = \mathcal{V}$. Moreover, the size of a subdomain is limited by the length of the bitsring b.

In theory, there is no limit on the number of subdomains and subdomains may overlap. However, more subdomains imply more forwarding information on the BFRs, more complex bitstring definition for a multicast group, and longer subdomain identifiers. Therefore, we keep the number of subdomains as low as possible. We further assume that any node of the BIER domain is a BFER. Therefore, the number of subdomains is $s = \lceil \frac{n}{b} \rceil$. We require the subdomains to be disjoint. This simplifies the algorithms and has no impact on the results as the network sizes in our experiments are multiples of maximum subdomain sizes.

2) Objective Function: The objective function for the clustering is to keep the overall traffic low. We define the overall traffic as the number of packets carried over all links of the BIER domain when every node sends a packet to every other node. The traffic induced by a BIER packet sent from a single BFIR v to all BFERs within a subdomain $\mathcal{S}$ is the number of edges traversed, i.e., $\left|\bigcup_{w\in\mathcal{S}} p(v, w)\right|$. Thus, the overall BIER traffic load is

$$\rho = \sum_{v\in\mathcal{V}} \sum_{\mathcal{S}\in\mathcal{C}} \left| \bigcup_{w\in\mathcal{S}} p(v, w). \right| \tag{1}$$

This metric is to be minimized by a clustering $\mathcal{C}$.

3) Alternate Optimization Goals: Future work may optimize the clustering so that more subdomains are allowed. Then, some subdomains may overlap and BFIRs can choose over which subdomains a BFER will be reached. This increases the potential for the minimization of overall traffic so that even fewer redundant BIER packets may need to be generated. This, however, requires the knowledge of the IPMC groups and needs more resources on BFRs and BFIRs.

B. Random BIER Clustering

We briefly explain random BIER clustering. A bitstring length of b is given. A set of n BFERs is subdivided into equal-size $s = \lceil \frac{n}{b} \rceil$ subdomains. BFERs are randomly assigned to these subdomains whereby their size is limited to b BFERs. In Section V-C we use this algorithm as a baseline for comparison.

C. Optimal BIER Clustering for Selected Topologies

We describe optimal clusterings for selected, regular topologies: full mesh, line, ring, and perfect binary tree. We renounce on a formal proof of optimality as this is rather obvious. For arbitrary topologies we provide an ILP-based optimization algorithm in the next subsection.

1) Full Mesh: Here, random assignment is optimal. In full meshes, all traffic is exchanged over a direct link between source and destination because all nodes are neighbors. However, in such topologies, there is no traffic reduction potential for multicast and we do not consider full meshes any further.

2) Line Topologies: Start at one end of the line. Assign the next b neighboring nodes to a subdomain. Repeat until all nodes are assigned. The last subdomain may have less than b nodes.

3) Ring Topologies: Select an arbitrary position in the ring and choose a direction. Assign the next b neighboring nodes to a subdomain. Repeat until all nodes are assigned. The last subdomain may have less than b nodes.

4) Perfect Binary Trees: We consider a perfect binary tree. The depth of a node is its distance to the root plus one so that the leaves have maximum depth. We denote their depth as the height h of the tree. We state that a perfect binary tree with height h has $2^h - 1$ nodes.

We assume that the bitstring size is $b = 2^k$. It can accommodate a perfect binary tree with height k. We give an algorithm to cluster a perfect binary tree with height h into 2^{h-k} subdomains with up to 2^k nodes. We take all subtrees with roots of depth $h-k+1$ as initial subdomains. The other unassigned nodes are assigned to a nearest possible subdomain which still accepts additional nodes. Thereby, the assignment order of these nodes is inverse to their depth. The order among nodes with equal depth does not matter.

D. Optimal BIER Clustering for Arbitrary Topologies

We first explain fundamentals of integer linear programs (ILPs). Then, we apply them for optimal clustering of BIER domains.

1) Fundamentals of ILPs: An ILP describes the solution space of an optimization problem with so-called decision variables and linear inequalities. Parameters of the optimization problem serve as coefficients in the inequalities. A linear objective function describes the quality of possible solutions and is to be minimized.

ILP solvers find the best integer solution for decision variables that fulfill all inequalities. During the solution process, an ILP solver indicates lower and upper bounds regarding the objective value for the best solution. The upper bound is the value for the best solution found so far. While progressing, better solutions may be found and the lower bound for the best solution may increase. If upper and lower bound meet, the ILP solver found an optimal solution.

2) BIER Clustering Using ILPs: We build an ILP that describes the solution space for BIER clustering and an objective function for the overall traffic load given in Equation (1). Its output is an optimal clustering $\mathcal{C}$ of the network that minimizes the objective function.

$$\forall v \in \mathcal{V}: \sum_{\mathcal{S}\in\mathcal{C}} x_v^{\mathcal{S}} = 1 \quad (2)$$

$$\mathcal{S} \in \mathcal{C}: \sum_{v\in\mathcal{V}} x_v^{\mathcal{S}} \le b \quad (3)$$

$$\forall v, w \in \mathcal{V}, e \in \mathcal{E}, \mathcal{S} \in \mathcal{C}: p_{e,v,w} \cdot x_w^{\mathcal{S}} \le y_{v,e}^{\mathcal{S}} \quad (4)$$

$$\forall v \in \mathcal{V}, e \in \mathcal{E}, \mathcal{S} \in \mathcal{C}: y_{v,e}^{\mathcal{S}} \le \sum_{w\in\mathcal{V}} p_{e,v,w} \cdot x_w^{\mathcal{S}} \quad (5)$$

$$\text{min: } \rho = \sum_{v\in\mathcal{V}} \sum_{\mathcal{S}\in\mathcal{C}} \sum_{e\in\mathcal{E}} y_{v,e}^{\mathcal{S}} \quad (6)$$

The ILP is given by Equation (2), Inequalities (3)–(5), and the objective function (6). It contains two types of binary decision variables. The decision variable $x_v^{\mathcal{S}}$ indicates whether node v belongs to subdomain $\mathcal{S}$; it is 1 if $v \in \mathcal{V}$ is in subdomain $\mathcal{S}$, otherwise it is 0. Equation 2 enforces that any node is part of exactly one subdomain. Inequaltiy 3 ensures that a subdomain contains at most b nodes. The decision variable $y_{v,e}^{\mathcal{S}}$ indicates whether edge e is part of the multicast tree from node v to any node $w \in \mathcal{S}$. It depends on $x_v^{\mathcal{S}}$ and the forwarding information. The latter is given by coefficients $p_{e,v,w}$ which are 1 if edge e is on the path from v to w; otherwise the coefficient is 0. This dependency is modelled by Inequalities 4 and 5. Equation 4 ensures that $y_{v,e}^{\mathcal{S}} = 1$ if e is part of the path from BFIR v to any BFER w in subdomain $\mathcal{S}$. Equation 5 ensures that the decision variable $y_{v,e}^{\mathcal{S}}$ is 0 if e is not part of any path from v (BFIR) to any w (BFER) in $\mathcal{S}$; thereby the membership $w \in \mathcal{S}$ is expressed only indirectly by $w \in \mathcal{V}$ and the decision variable $x_w^{\mathcal{S}}$.

The objective function in Equation (6) quantifies the overall traffic as defined in Equation (1) and is to be minimized.

E. Heuristic BIER Clustering

We propose a heuristic clustering algorithm that consists of two phases. Phase 1 selects initial subdomains. Phase 2 improves these subdomains according to Equation (1) by exchanging the assignment of node pairs to their subdomains.

Phase 1 works as follows. First, randomly select s nodes as center nodes of the different subdomains. Second, add further nodes to the subdomains until their maximum size b is reached. To that end, nearest non-assigned nodes are assigned to the center nodes in round-robin fashion. This yields a clustering of the BIER domain into subdomains. We repeat Phase 1 to generate $10 \cdot s$ clusterings and choose the best according to the objective function in Equation (1) to continue with it in Phase 2.

Phase 2 improves the clustering. First, randomly select two nodes that have neighbors in other subdomains and that are are assigned to different subdomains. Swap their assignment if this reduces the overall load according to Equation (1). Repeat this procedure until ρ from Equation (1) does not decrease for $n = |\mathcal{V}|$ steps. When computing a clustering for a network, we perform the presented algorithm 20 times and take the best solution.

This algorithm is simple but works better than more complex approaches we have evaluated before. We evaluate the quality of this heuristic in the next section.

V. Comparison of BIER Clustering Algorithms

In this section we compare the BIER clustering algorithms from the previous section with regard to runtime and quality. First we present the topologies that we use for evaluations in this paper. Then, we demonstrate that the runtime of the ILP-based optimization is feasible only for small networks. Finally, we compare the quality of the subdomains obtained for different algorithms, topologies, and network sizes.

A. Topologies

In this work we investigate delivery of multicast traffic in various network topologies: full mesh, line, ring, perfect binary tree, and mesh networks with node degree $d \in \{2, 4, 6, 8\}$. We refer to the latter as mesh-d. We construct them using the topology generator BRITE [37] which leverages a Waxman model [38]. While the first mentioned topologies are regular so that there is only a single choice for a network with n nodes, mesh-d networks are randomly constructed. Therefore, we generate 10 different representatives and compute average values for the considered metrics. The 95% confidence intervals are below 0.3% for all reported results so that we omit them in all tables and figures.

Topology	$n = 64$		$n = 128$	
	$s = 2$	$s = 4$	$s = 2$	$s = 4$
Line	0.11	3.80	1.07	45.51
Ring	66.51	21139.70	3633.59	-
Perfect binary tree	0.11	1.10	0.33	6.71
Mesh-2	0.06	3.59	0.21	22.67
Mesh-4	76.09	-	-	-
Mesh-6	718.23	-	-	-
Mesh-8	3883.62	-	-	-

Tab. 1: Time to solve ILPs for BIER clustering in seconds. Some instances could not be solved within 12 hours.

B. Runtime for ILP-Based Optimization

We measure the runtime to solve ILPs for BIER clustering with the ILP solver Gurobi 9.1 on a Ryzen 3900X CPU with 12 cores running at 3.8 GHz with 64 GB RAM.

Table 1 compiles the runtimes of the solver for different network topologies, network sizes, and number of subdomains. Perfect binary trees have one node less than indicated in the table. The runtime to solve the ILPs increases with network size and in particular with the number of subdomains. The network topology also has a significant impact. For some topologies, networks with 128 nodes or with 4 subdomains cannot be solved within 12 hours.

In contrast, the heuristic algorithm has a runtime of a few seconds for any topology with $n = 1024$ nodes, and $s = 4$ subdomains. For $n = 8192$ nodes and $s = 32$ subdomains, it takes 8–16 h for mesh-4 and mesh-6, 16–24 h for lines, perfect binary trees, mesh-2, and mesh-8, and about 3 days for rings.

Thus, solving the ILP for optimal BIER clustering is infeasible for realistic problem instances, but the runtime of the heuristic algorithm is acceptable even for large networks. Therefore, we utilize for the evaluations in Section VI the topology-specific solutions of Section IV-C for lines, rings, and perfect binary trees, and the heuristic algorithm for mesh-d networks.

C. Quality Comparison

We now compare the quality of heuristic results with those from optimal and random subdomain assignment. The metric is the overall traffic load ρ with BIER when every node sends a packet to any other node (see Equation (1)).

We first consider mesh-d, for which only the ILP-based algorithm can deliver optimal results but only for small networks. Table 2 shows the overall traffic for subdomains generated with heuristic and with random assignment relative to the overall traffic for optimal subdomains. All heuristic results are close to optimum. We observe for mesh-2 that larger networks and more subdomains slightly degrade the results of the heuristic algorithm. Random assignment is clearly worse, i.e., it generates 33%-80% more extra traffic than optimal subdomains while heuristic assignment causes only 0.3%-1.5% more extra traffic. The quality of the heuristic results tends to improve with increasing node degree.

Topology	n	s	Heuristic (%)	Random (%)
Mesh-2	64	2	100.3	132.6
		4	100.7	162.2
	128	2	100.5	133.7
		4	101.5	179.8
Mesh-4	64	2	100.3	115.2
Mesh-6	64	2	100.4	110.6
Mesh-8	64	2	100.3	107.1

Tab. 2: Overall traffic load for heuristic and random BIER clustering depending on network size n and number of subdomains s; numbers are relative to the overall traffic load for optimal subdomains computed based on ILP solutions.

Now we discuss larger, regular topologies for which the algorithms of Section IV-C provide optimal results. We cluster the networks into subdomains of size $b = 256$. The results are compiled in Table 3. We consider networks with $n \in \{256, 512, 1024, 2048, 4096, 8192\}$ nodes, an exception are perfect binary trees with only $n - 1$ nodes. Consequently, multiple subdomains $s \in \{1, 2, 4, 8, 16, 32\}$ are needed. The overall traffic load is given relative to the one for optimal subdomains.

n	s	Line (%)		Ring (%)		Perfect binary tree (%)	
		Heur.	Rnd.	Heur.	Rnd.	Heur.	Rnd.
256	1	100	100	100	100	100	100
512	2	100	159.5	100	133.1	101.2	142.8
1024	4	100	212.2	100	199.1	100.8	197.2
2048	8	100	249.3	100	265.1	100.6	262.5
4096	16	100	271.8	108.7	317.9	104.9	336.9
8192	32	100	284.3	134.1	353.0	118.0	416.9

Tab. 3: Overall traffic load for heuristic and random BIER clustering depending on network size n and number of subdomains s; numbers are relative to the overall traffic load for optimal subdomains computed based on topology-specific solutions.

We observe that the quality of the heuristic is almost optimal for up to 2048 nodes. Beyond that, the quality degrades by up to 34% for rings compared to optimum. The quality for lines and perfect binary trees is better with a degradation of at most 18%. The results with random assignment are much worse than those with optimum and heuristic assignment.

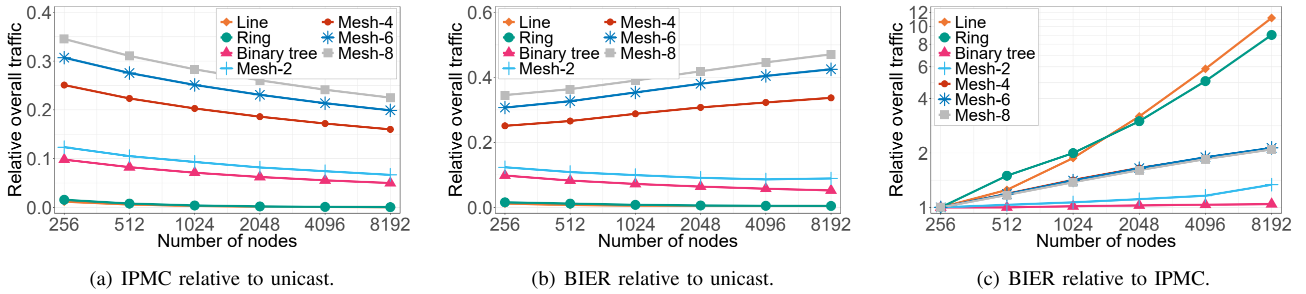
We draw two major conclusions. First, optimization of subdomains is important as random subdomains are likely to cause a lot more extra traffic than needed in large BIER domains. Second, subdomains obtained through the presented heuristic are almost optimal for networks up to 2048 nodes, beyond that we see a degradation. However, even then heuristic subdomains are still much better than random subdomains. The heuristic is needed for the evaluation of mesh-d networks in Section VI. We believe that the quality of the heuristic results for mesh-d is acceptable even for large networks because the heuristic algorithm performed well in large networks for lines, rings, and perfect binary trees. Therefore, the method may be suitable for application in practice.

VI. Traffic Savings with IPMC and BIER

In this section we investigate the potential of multicast variants, i.e., IPMC and BIER, to reduce the traffic load from multicast traffic relative to unicast, and compare it with each other. We first discuss the methodology. Afterwards we study the reduction potential for overall traffic depending on network size and multicast group size. Then, we show that both IPMC and BIER can well avoid heavily loaded links. Finally, we examine the impact of header size on the traffic saving potential of BIER.

A. Methodology

We describe the general approach, investigated network topologies, and how BIER subdomains are clustered.



(a) IPMC relative to unicast. (b) BIER relative to unicast. (c) BIER relative to IPMC.

Fig. 3: Overall traffic load when every node sends one packet to all other nodes, depending on the network size.

1) General Approach: It is obvious that multicast groups can be very different, both in size and geographical distribution. Moreover, networks supporting multicast can have different topology. As those factors likely impact the efficiency of multicast variants, we study them depending on network topology, network size, and multicast group size. We study the topologies presented in Section V-A; if the topologies are random, we report averages from 10 different topologies and omit the small confidence intervals as mentioned. The networks have $n \in \{256, 512, 1024, 2048, 4096, 8192\}$ nodes, with the exception of perfect binary trees that have only network size $n-1$. To quantify the traffic savings potential of IPMC and BIER, we relate their overall load for a specific traffic scenario to the one of unicast. The overall load is the number of packets resulting from the traffic scenario, summed up over all links in the network (cf. Equation (1) for BIER). The considered traffic scenarios are maximum multicast groups, i.e., every node sends a packet to all other nodes in the network, and partial multicast groups, i.e., every nodes sends a packet to a given number of randomly chosen nodes. For all evaluations, packets follow shortest path trees based on the hop count metric.

The traffic reduction potential may vary among links. Central links tend to have a much higher load than others so that they may profit more from traffic load reduction through IPMC or BIER. Therefore, we also study link load reduction on links.

2) BIER Clustering: If not stated otherwise, we assume in our studies for BIER a bitstring size of $b = 256$ bits because that value must be supported by all BIER-capable equipment. Thus, b is also the maximum number of BFERs in subdomains. We assume that all nodes are BFERs. That means, when networks have more than b nodes, the nodes are partitioned into a minimum number of $s = \lceil \frac{n}{b} \rceil$ subdomains.

For lines, rings, and perfect binary trees, the optimum clustering from Section IV-C is applied while random mesh topologies are clustered using the heuristic algorithm from Section IV-E.

B. Reduction of Overall Traffic

We evaluate the potential for the reduction of overall traffic through multicast variants relative to unicast and compare the efficiency of BIER with the one of IPMC. We first study the impact of network topology and size and then the impact of network toplogy and multicast group size.

1) Impact of Network Size: We evaluate the savings potential for overall traffic through multicast variants. To that end, we consider different network topologies and sizes and maximum multicast groups. We study IPMC vs. unicast, BIER vs. unicast, and BIER vs. IPMC.

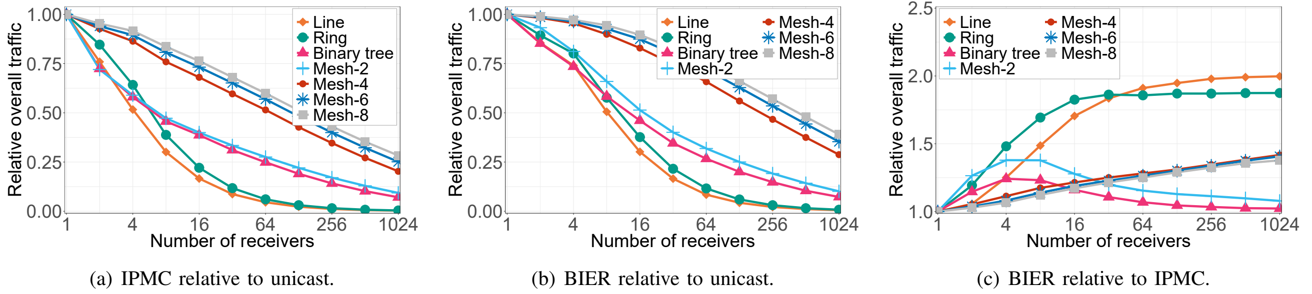
a) IPMC vs. Unicast: Figure 3(a) shows the overall traffic for IPMC relative to unicast for multiple network topologies depending on the network size. The IPMC traffic load decreases relative to the unicast traffic load with increasing network size. There is a large reduction potential in line and ring networks so that the IPMC traffic volume is less than 2% compared to the one of unicast. In perfect binary trees the traffic can be reduced to 10% for $n = 255$ nodes and to 5% for $n = 8191$ nodes. Random mesh networks have a lower reduction potential that decreases with increasing node degree.

We observe an obvious dependence of the traffic reduction potential of IPMC on the network topology. We show that it is $\frac{1}{l}$ in the presence of maximum multicast groups. Multicast requires $n-1$ hops to distribute a packet from one source to $n-1$ receivers as this is the number of links in any shortest-path tree. Thus, $n \cdot (n-1)$ hops are required to distribute a packet from each node to all other nodes. When the same is done with unicast, any source node $v \in \mathcal{V}$ sends a packet to any destination node $w \in \mathcal{V}$. This requires $|p(v, w)|$ hops per v/w pair, which is in sum

$$\sum_{v\in\mathcal{V}}\sum_{w\in\mathcal{V}} |p(v,w)| = n\cdot(n-1)\cdot \frac{\sum_{v\in\mathcal{V}}\sum_{w\in\mathcal{V}} |p(v,w)|}{n\cdot(n-1)} = n\cdot(n-1)\cdot l. \quad (7)$$

This follows that IPMC can reduce the overall traffic to $\frac{1}{l}$ compared to unicast. Lines and rings have by far the longest average path length and it strongly increases with increasing network size. In other topologies, average path lengths are clearly lower and increase slowly with the network size. The average path length correlates with the node degree and increases in the following topologies: mesh-8, mesh-6, mesh-4, mesh-2 and perfect binary trees.

b) BIER vs. Unicast: Figure 3(b) presents the overall traffic load for BIER relative to unicast. The number of required subdomains increases with the network size and thereby



(a) IPMC relative to unicast.

(b) BIER relative to unicast.

(c) BIER relative to IPMC.

Fig. 4: Overall traffic load depending on the size of the multicast group in networks with $n = 1024$ nodes.

the number of BIER packets needed for these subdomains. We observe that BIER achieves the largest traffic reduction in lines and rings (around 97%), followed by perfect binary trees and mesh networks with node degree 2 (around 90%). Mesh networks with node degrees 4, 6, and 8 have a lower savings potential of around 50% and BIER's traffic savings potential relative to unicast decreases with increasing network size. The latter is different to IPMC (cf. Figure 3(a)). Thus, BIER can reduce the load of multicast traffic similarly well as IPMC, except in large, highly meshed networks in spite of well clustered subdomains.

c) BIER vs. IPMC: To compare BIER directly with IPMC, we consider the fraction of overall traffic load of BIER and the one of IPMC in Figure 3(c). In line and ring networks, BIER causes a multiple (3 - 11 times) of the traffic that occurs with IPMC if the number of subdomains is very large, i.e., 8, 16, and 32 for network sizes of 2048, 4096, and 8192 nodes. However, the savings compared to unicast are still enormous, i.e., more than 98%. In mesh networks with node degee 4, 6, and 8, the traffic load with BIER compared to IPMC increases about logarithmically with network the network size and roughly doubles the load with IPMC in very large networks. In perfect binary trees and mesh networks with node degree 2, the traffic load with BIER relative to IPMC also increases with network size, but BIER causes only 40% more traffic than IPMC in very large networks although up to 32 subdomains are supported.

2) Impact of Multicast Group Size: We evaluate the influence of the multicast group size on the traffic reduction potential of multicast variants. We perform this study for different network topologies and a network size of $n = 1024$. Every node sends a packet to a random set of receivers. The set sizes are $r \in \{1, 3, 7, 15, 31, 63, 127, 255, 511, 1023\}$ large, for perfect binary trees the maximum number of receivers is $r = 1022$. As these are random experiments, we run each experiment 20 times to obtain very small confidence intervals that we omit in the figures for the sake of clarity.

a) IPMC vs. Unicast: Figure 4(a) shows the overall traffic load of IPMC relative to unicast. It decreases with increasing multicast group size. For small multicast group sizes, IPMC can reduce the overall traffic only by little. In line and ring networks, large traffic savings are achieved already for small multicast groups, followed by perfect binary trees and random meshes with node degree 2. Random meshes with node degree 4, 6, and 8 require rather large multicast groups to provide a substantial savings potential.

b) BIER vs. Unicast: Figure 4(b) shows the overall traffic load for BIER vs. unicast. The figure looks similar to Figure 4(a), but all lines are slightly higher due to the extra traffic caused with BIER. To make the difference between BIER and unicast better visible, we compare their results directly in the following.

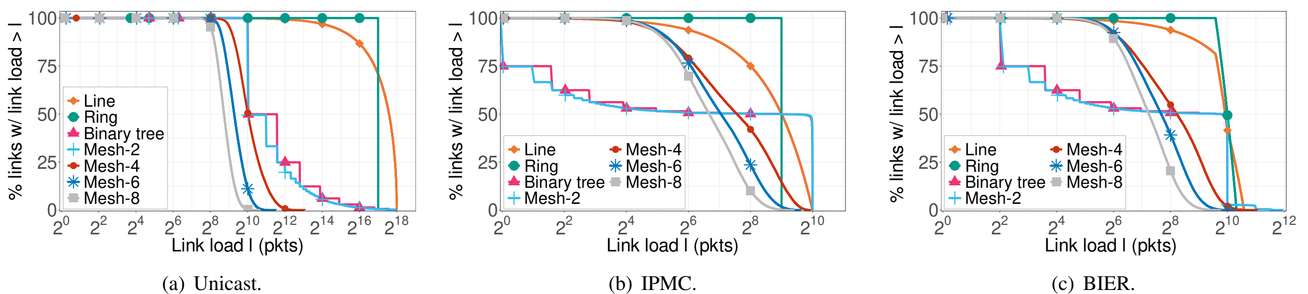
c) BIER vs. IPMC: Figure 4(c) shows the overall traffic load with BIER compared to the one of IPMC. For a single receiver, BIER and IPMC are equally efficient as BIER does not send any redundant packets. When the number of receivers increases, the overhead of BIER increases as more redundant BIER copies are sent. The values for $r = 1024$ receivers are same as the values for network size $n = 1024$ in Figure 3(c). Apart from that, BIER's overhead compared to IPMC depends on the network topology. For line and ring topologies, the overhead of BIER relative to IPMC increases up to a multicast group size of 64 receivers and remains constant afterwards. For mesh networks with node degree 4, 6, and 8, the overhead increases logarithmically with increasing number of subscribers. And for line and ring networks, the overhead decreases when the number of subscribers is 8 receivers or more.

C. Avoidance of Heavily Loaded Links

For some network topologies, the savings potential through multicast is only moderate. However, traffic is not equally distributed over all links of a network as central links tend to carry more traffic. We show that multicast variants can greatly reduce the load on those links compared to unicast.

We consider again maximum multicast groups and networks with $n = 1024$ nodes, $n = 1023$ for perfect binary trees. We count the number of packets carried over each link and discuss the complementary cumulative distribution function (CCDF) of the link loads for unicast, IPMC, and BIER. In case of mesh-d, the load information of multiple networks is integrated in a single CCDF.

1) Link Load Distribution with Unicast: The CCDF for link loads with unicast is illustrated in Figure 5(a). In line and ring networks, a large percentage of links carries a large number of packets. With rings, any link carries the same number of packets due to symmetry. In perfect binary trees the number of



(a) Unicast.

(b) IPMC.

(c) BIER.

Fig. 5: Complementary cumulative distribution function (CCDF) for link loads in networks with 1024 nodes; every node sends a packet to all other nodes.

packets per link is clearly lower than in lines and rings. Half of the links has only a load of 1022 packets, those are adjacent to the leaves. There are also a few links with a very high link load, those are links close the root. Random mesh networks with a node degree 2 have a similar CCDF as perfect binary trees. The random mesh networks with a node degree of 4, 6, and 8 have increasingly lower link loads and less variation regarding link loads.

2) Link Load Distribution with IPMC: Figure 5(b) shows the CCDF for link load with IPMC. The upper limit is now 1023 packets. Again, many links in lines and rings carry a large number of packets. In perfect binary trees and random mesh networks with node degree 2, 25% of the links have a very low load while 50% have a high load. In perfect binary trees, those are links close to the leaves and to the root, respectively. Random meshes with node degree 4, 6, and 8 reveal again a load continuum but it is at a lower load level compared to unicast. The x-scales in Figures 5(a) and 5(b) are different. This suggests that there are many links for which IPMC decreases the traffic load by orders of magnitude. Thus, IPMC avoids in particular heavily loaded links compared to unicast.

3) Link Load Distribution with BIER: Figure 5(c) shows the CCDF of link loads with BIER. It looks similar to the one of IPMC in that the link load is mostly limited to 1023 packets. However, some links in line and ring networks have larger loads up to around 1600 packets, and a very few links in mesh networks with node degree 2 have loads of up to 3976 packets, i.e., roughly the maximum link load for multicast multiplied by the number of subdomains s. Mesh networks with a node degree of 4, 6, and 8 generally lead to lower link loads as their traffic is not concentrated on a few links.

The most important finding is that BIER also avoids very high loads on links compared to unicast. Only a very few links experience substantially higher link loads than with IPMC. Thus, BIER efficiently avoids heavily loaded links in a similar way as IPMC.

D. Impact of BIER Header Size

On the one hand, BIER largely avoids redundant packets over links, on the other hand BIER causes additional header overhead. There is an obvious tradeoff regarding header size:

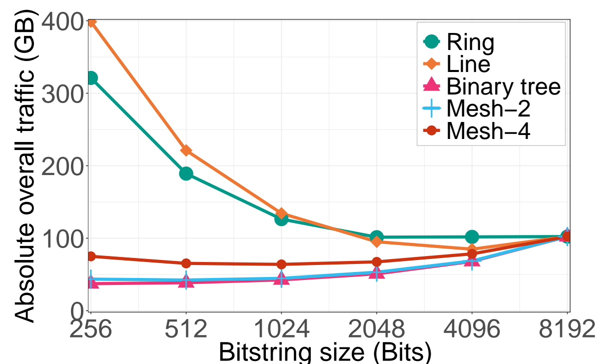


Fig. 6: Overall traffic load depending on the bitstring size in the BIER header in networks with $n = 8192$ nodes.

small bitstrings add only little header overhead, but require many redundant packets in large network, large bitstrings add lots of header overhead, but require only a few redundant packets in large networks. We expect an optimal header size in between. To study this effect, we first explain our methodology and then discuss experimental results.

1) Methodology: We consider networks with $n = 8192$ nodes, $n = 8191$ for perfect binary trees. Multicast groups have maximum size, i.e., n participants, and each participant sends a single packet. We investigate different bitstring sizes of $b \in \{256, 512, 1024, 2048, 4096, 8192\}$ bits that require $\lceil \frac{n}{b} \rceil$ SDs. We assume a payload size of 500 B which is the average size of IP packets on the Internet [39]. In contrast to the preceding experiments, we now take the amount of overall traffic in bytes as performance metric as this captures the effect of the BIER header size. We omit mesh-6 and mesh-8 topologies because hop count and average path lengths are almost identical to mesh-4.

2) Results: Figure 6 shows the overall traffic volume for different bitstring sizes. The ring and the line topology lead to a large traffic volume for small bitstring sizes b. This results from long paths of most of the packets replicated for the $\lceil \frac{n}{b} \rceil$ SDs. Larger bitstring sizes reduce the number of SDs and thereby the replicated packets as well as the traffic volume. The optimal bitstring size for the line is $b = 4096$ bits and

the one for the ring is $b = 2048$ bits. Larger bitstring sizes add so much header overhead that the overall traffic increases again. Topologies with shorter paths like binary trees, mesh-2 or mesh-4 networks reveal a clearly lower traffic volume for small bitstring sizes than lines or rings. The optimal bitstring size is $b = 256$ bits for binary trees, it is $b = 512$ bits for mesh-2, and it is $b = 1024$ bits for mesh-4. However, suboptimal bitstring sizes between 256 and 2048 bits lead only to slightly larger traffic volumes. Thus, any of these bitstring sizes is suitable for typical network topologies.

VII. Impact of Single Link Failures

We have optimized BIER subdomains for failure-free forwarding. In case of link failures, rerouting occurs in IP networks and then traffic is diverted around failed links. As a consequence, individual link loads and overall traffic load may increase. BIER with subdomains optimized for failure-free routing may lead to an even larger traffic increase than IPMC forwarding. Therefore, BIER may require more backup capacity than IPMC. We investigate these issues in the following. We first explain our methodology. Then, we study the overall traffic load and maximum link loads in case of single link failures, as well as the overall backup capacity required to accommodate rerouted traffic.

A. Methodology

Single link failures may partition a network topology. Then multicast groups are also partitioned into subgroups that cannot reach each other anymore. This can be avoided in resilient networks with 2-link-connected topologies and rerouting after failure detection. Thereby, end-to-end connectivity is not impaired so that participants of a multicast group can still reach each other. As a consequence, we consider only 2-link-connected topologies in this context, i.e., networks which are still connected after any single link failure. As a consequence, we do not consider lines and binary trees as they may be partitioned through single link failures. Rings are 2-link-connected by definition. We reuse the mesh-{4,6,8} toplogies from Section V-A which were chosen for the entire study such that they are 2-link-connected.

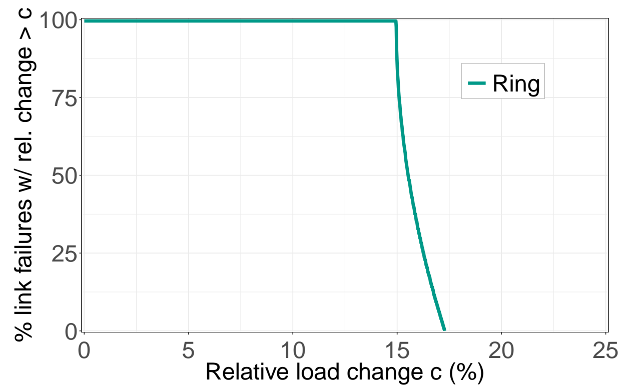
We consider networks with 1024 nodes and a bitstring with $b = 256$ bits. We optimize the subdomains for the failure-free case using the heuristic clustering algorithms from Section IV-E for mesh-d topologies, and the optimal clustering algorithm from Section IV-C for the ring topology. We assume again a full multicast group and each participant sends a single packet to all other participants. We compute the effect of all single link failures for the mentioned topologies. That means, we remove the failed link from the topology, calculate new shortest paths, and compute the performance metric based on the new traffic distribution; thereby, the subdomains remain unchanged. As mesh-d topologies are random, we report averaged results for them from 10 different topology samples.

In our experiments we count number of packets carried over links. When we extend the single sent packets to flows, we obtain observed rates which are proportional to the numbers of counted packets. To be more intuitive, we sometimes talk about rates and required capacities rather than counted packets, in particular when it comes to backup resources.

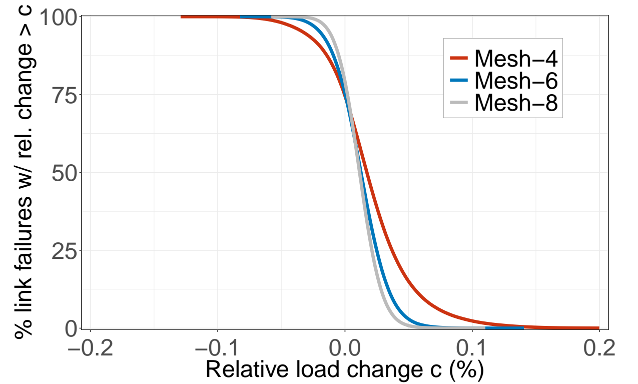
B. Overall Traffic Load

Traffic rerouting due to link failures possibly leads to longer paths, which may increase the overall link load in the network. Thus, we quantify the impact of single link failures on the overall traffic load (Equation 1) and compare it to the failure-free case both for IPMC and for BIER.

As the multicast groups in our experiments contain every node in the network, the overall traffic load for IPMC is $n \cdot (n-1)$ packets, no matter if a link fails. This is due to the fact that n packets are each forwarded along a single shortest path tree, and each shortest path tree consists of $n-1$ hops. Thus, the traffic load does not increase with IPMC in case of single link failures.



(a) Ring.



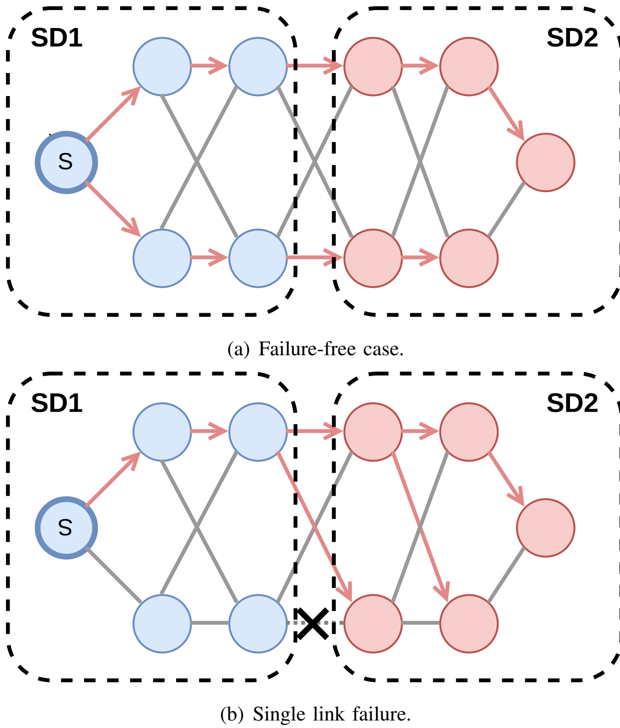
(b) Mesh-d.

Fig. 7: BIER with single link failures – CCDFs of relative overall traffic change compared to the failure-free case, accumulated over all single link failures. Experiments are conducted in networks with $n = 1024$ nodes, every node sends a packet to every other node.

This is different with BIER. With BIER, $\lceil \frac{1024}{256} \rceil = 4$ packet copies, one for each subdomain, are forwarded over shortest path trees which consist of fewer hops than $n-1$.

However, their overall number of hops may change when traffic is rerouted. Therefore, we evaluate the change of overall traffic load with BIER for all single link failures. Figures 7(a) and 7(b) show CCDFs of relative overall traffic changes accumulated over all single link failures. We first discuss Figure 7(a) for a ring network. The overall traffic load rises between 15% and 17.3% depending on the position of the failed link. We explain this large increase as follows. Between any two nodes, there are exactly two paths in a ring network and the paths may have significantly different length. If the shorter path fails, traffic is rerouted over the longer path. This causes path stretch and leads to the observed increase in overall traffic load.

We now study mesh-d topologies for which the CCDF of the change in overall traffic load is presented in Figure 7(b). The increase in overall traffic load is bounded by 0.2%. We explain this as follows. In meshed networks with a node degree between 4 and 8, multiple paths exist between any two nodes and their lengths are likely to be similar. If the shortest path fails, another path with similar length is mostly available, which hardly increases the overall traffic load.



(a) Failure-free case.

(b) Single link failure.

Fig. 8: Example network with two BIER subdomains. In case of the indicated link failure, the adapted shortest path tree for the nodes in SD2 contains fewer hops than in the failure-free case, which reduces the traffic load.

We further observe that in 75% of all single link failures, the overall traffic load increases but in 25% the overall load decreases. This observation does not seem intuitive as the shortest path length for any pair of nodes remains unchanged or increases in case of a link failure. Nevertheless, the load may decrease as the shortest path tree towards the nodes in a subdomain may be more compact after rerouting. We illustrate this claim by the example in Figures 8(a) and 8(b). They show a network partitioned into two subdomains, SD1 and SD2. The shortest path tree starting in node S towards all nodes in SD2 contains two hops less in case of the considered link failure (Figure 8(b)) than under failure-free conditions (Figure 8(a)). This apparently more favorable path layout cannot be utilized under failure-free conditions because BIER traffic is always forwarded according to the paths in the underlay.

C. Maximum Load Increase on Links

When traffic is rerouted over another path, the traffic load on the corresponding links increases. We record for each link the maximum load increase observed for any single link failure as this constitutes the required backup capacity for this link.

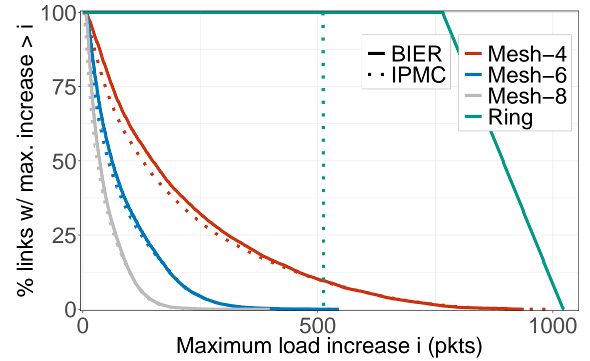
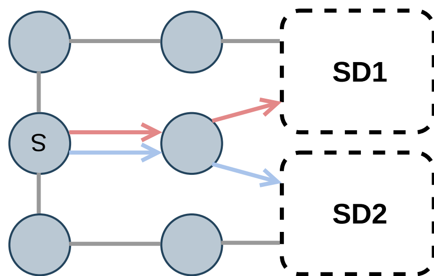
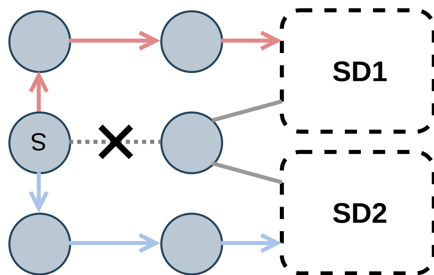


Fig. 9: CCDFs of maximum link load increases for single link failures. Experiments are conducted in networks with $n = 1024$ nodes, every node sends a packet to every other node.

Figure 9 shows the CCDFs of the maximum load increases for all links. In ring networks, all links experience up to 512 more packets with IPMC in case of link failures. In contrast with BIER, links carry between 768 and 1024 more packets. This is because multiple redundant BIER packets may be affected by the failure and are redirected. Therefore, BIER requires substantially more backup capacity in rings than IPMC and the exact amount depends on the location of a link within its subdomain. In mesh-d networks, the CCDF is almost a continuum. In networks with larger node degree, links require less backup than in networks with smaller node degree. This is due to shorter paths and less affected traffic, shorter backup paths, and better traffic distribution in case of link failures. Most notably, BIER causes about the same maximum load increases as IPMC although BIER requires more capacity than IPMC under failure-free conditions. We explain this fact by an example. Figure 10(a) shows a link carrying redundant BIER packets to two different subdomains. When that link fails, the traffic is redirected over different paths to the subdomains. IPMC would save a packet copy in the failure-free case, but it results into the same traffic distribution in this particular example.



(a) Failure-free case: two redundant packets are delivered over a link to two different subdomains.



(b) Single link failure: the two packets are redirected over different backup paths.

Fig. 10: Example network with two BIER subdomains. Redundant BIER packets for different subdomains are redirected over different paths.

	Metric	Ring	Mesh-4	Mesh-6	Mesh-8
IPMC	Cap. w/o backup	1047552	1047552	1047552	1047552
	Cap. w/ backup	2095104	1857633	1559138	1422539
	Abs. backup cap.	1047552	810081	511586	374987
	Rel. backup cap.	1.00	0.77	0.49	0.36
BIER	Cap. w/o backup	1051129	1395817	1418709	1406915
	Cap. w/ backup	2881534	2263645	1962557	1813694
	Abs. backup cap.	1830405	867828	543848	406779
	Rel. backup cap.	1.74	0.62	0.38	0.29
BIER / IPMC	Fraction w/o backup	1.003	1.33	1.35	1.34
	Fraction w/ backup	1.375	1.22	1.26	1.27

Tab. 4: Overall capacity w/ and w/o backup as well as absolute and relative backup capacity for IPMC and BIER. Capacities are given in packets.

D. Overall Backup Capacity

We sum up link capacities for a network needed to carry the considered traffic for failure-free conditions on the one hand (capacity w/o backup) and for all single link failures on the other hand (capacity w/ backup). The difference is the absolute backup capacity. Table 4 compiles them for BIER and IPMC in mesh-d and ring topologies. The relative backup capacity is the ratio between absolute backup capacity and capacity w/o backup.

The results show that IPMC require 100% relative backup capacities for rings, but only 77%, 49%, and 36% for mesh-4, mesh-6, and mesh-8 networks. In contrast, BIER needs 176% backup capacity for rings, and 62%, 38%, and 29% for mesh-4, mesh-6, and mesh-8 networks. This is less than for IPMC, which seems surprising, but there is a simple explanation. BIER requires more capacity w/o backup than IPMC, but only little more backup capacity than IPMC. As a consequence, BIER's relative backup capacity is lower than the one for IPMC.

Below the line, BIER does not lead to excessive backup capacity demands when BIER subdomains are optimized for failure-free scenarios. The relative backup capacity is even lower than with IPMC. The ring network is an exception, but also IPMC requires lots of capacity in rings.

VIII. Conclusion

BIER is a novel forwarding paradigm to carry IP multicast (IPMC) traffic within so-called BIER domains. It is more scalable than IPMC because core nodes remain unaware of individual multicast groups. A problem arises for large BIER domains where subdomains need to be defined to make all egress nodes addressable. When an IPMC packet is distributed via a BIER domain, a separate BIER packet is needed for each subdomain that has a receiver for the IPMC packet. This leads to redundant packets and we showed that their number almost equals the number of subdomains if multicast groups are about 3 times larger than the number of subdomains. These redundant packets can significantly degrade BIER's ability to efficiently carry multicast traffic.

We argued that an appropriate choice of the subdomains can mitigate that effect when multiple BIER packets are sent to different regions of a network. Therefore, we defined the BIER clustering problem and proposed several algorithms to cluster a BIER domain into appropriate subdomains. We compared the runtime and quality of these algorithms, and showed that optimization of subdomains can greatly reduce the resulting overall traffic compared to random subdomains.

We evaluated and compared the ability of IPMC and BIER to reduce traffic load for multicast traffic relative to unicast transmission in different network topologies. It depends on the average path length in the network. IPMC can save lots of traffic in line and ring networks, in binary trees and in mesh networks with a low node degree. In mesh networks with larger node degree the traffic savings potential is smaller. It also depends on the network size. As BIER possibly sends redundant packets in large domains, its ability to reduce traffic load diminishes compared to IPMC. This also depends on network topology and size. In large networks with 8192 nodes and subdomain sizes of 256 nodes, BIER causes only moderate extra traffic compared to IPMC in binary trees and mesh networks with small node degrees. In contrast, it produces 10-12 times more traffic than IPMC in lines and rings, but the traffic savings potential of BIER is still very large in

these topologies ($\sim$ 98%). In mesh networks with larger node degrees BIER doubles the overall traffic compared to IPMC and also the traffic savings potential is clearly reduced. These findings hold for maximum multicast groups. In smaller multicast groups the traffic savings potential of IPMC and BIER relative to unicast transmission is lower. While unicast causes enormous traffic loads on some links, both IPMC and BIER decrease such loads by orders of magnitude. The residual load on these links is higher with BIER than with IPMC due to redundant packets, but it is still on a low level. We showed that there is an optimum size for the BIER header which depends on the network topology. We investigated the impact of single link failures on BIER domains with optimized subdomains. Rerouting causes only little more traffic load and the backup capacity needed for BIER networks is only little more tha the one of pure IPMC networks. Below the line, subdomains are a good means to scale BIER to large networks, but they need to be carefully chosen to minimize extra traffic due to redundant packets.

Further studies may improve BIER clustering algorithms with regard to quality. They may also consider alternate optimization goals such as the ability to take advantage of overlapping subdomains for known multicast groups. Furthermore, scaling BIER-TE is a related problem but it requires different approaches.

REFERENCES

[1] I. Wijnands *et al.*, *RFC 8279: Multicast Using Bit Index Explicit Replication (BIER)*, https://datatracker.ietf.org/doc/rfc8279/, Nov. 2017.

[2] S. Islam *et al.*, “A Survey on Multicasting in Software-Defined Networking,” *IEEE Communications Surveys Tutorials (COMST)*, vol. 20, 2018.

[3] Z. Al-Saeed *et al.*, “Multicasting in Software Defined Networks: A Comprehensive Survey,” *Journal of Network and Computer Applications (JNCA)*, vol. 104, 2018.

[4] M. Shahbaz *et al.*, “Elmo: Source Routed Multicast for Public Clouds,” in *ACM SIGCOMM*, 2019.

[5] A. Iyer *et al.*, “Avalanche: Data Center Multicast using Software Defined Networking,” in *International Conference on Communication Systems and Networks*, 2014.

[6] W. Cui *et al.*, “Scalable and Load-Balanced Data Center Multicast,” in *IEEE GLOBECOM*, 2015.

[7] X. Zhang *et al.*, “A Centralized Optimization Solution for Application Layer Multicast Tree,” *IEEE Transactions on Network and Service Management (TNSM)*, vol. 14, 2017.

[8] K. Mokhtarian *et al.*, “Minimum-delay multicast algorithms for mesh overlays,” *IEEE/ACM Transactions on Networking*, vol. 23, 2015.

[9] X. Li *et al.*, “Scaling IP Multicast on Datacenter Topologies,” in *ACM CoNEXT*, 2013.

[10] M. A. Kaafar *et al.*, “A Locating-First Approach for Scalable Overlay Multicast,” in *IEEE INFOCOM*, 2006.

[11] J. Rückert *et al.*, “Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks,” *Journal of Network and Systems Management (JNSM)*, vol. 23, 2015.

[12] J. Rueckert *et al.*, “Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm,” *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, 2016.

[13] Y.-D. Lin *et al.*, “Scalable Multicasting with Multiple Shared Trees in Software Defined Networking,” *Journal of Network and Computer Applications (JNCA)*, vol. 78, 2017.

[14] M. J. Reed *et al.*, “Stateless Multicast Switching in Software Defined Networks,” in *IEEE International Conference on Communications (ICC)*, 2016.

[15] S.-H. Shen, “Efficient SVC Multicast Streaming for Video Conferencing With SDN Control,” *IEEE Transactions on Network and Service Management (TNSM)*, vol. 16, 2019.

[16] T. Humernbrum *et al.*, “Towards Efficient Multicast Communication in Software-Defined Networks,” in *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2016.

[17] W. K. Jia *et al.*, “A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks,” *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 31, 2013.

[18] C. A. S. Oliveira *et al.*, “Steiner Trees and Multicast,” *Mathematical Aspects of Network Routing Optimization*, vol. 53, 2011.

[19] L. H. Huang *et al.*, “Scalable and Bandwidth-Efficient Multicast for Software-Defined Networks,” in *IEEE GLOBECOM*, 2014.

[20] S. Zhou *et al.*, “Cost-Efficient and Scalable Multicast Tree in Software Defined Networking,” in *Algorithms and Architectures for Parallel Processing*, 2015.

[21] Z. Hu *et al.*, “Multicast Routing with Uncertain Sources in Software-Defined Network,” in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2016.

[22] J.-R. Jiang *et al.*, “Constructing Multiple Steiner Trees for Software-Defined Networking Multicast,” in *Conference on Future Internet Technologies*, 2016.

[23] B. Ren *et al.*, “The Packing Problem of Uncertain Multicasts,” *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.

[24] S.-H. Shen *et al.*, “Reliable Multicast Routing for Software-Defined Networks,” in *IEEE INFOCOM*, 2015.

[25] A. Giorgetti *et al.*, “First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting,” in *IEEE European Conference on Networks and Communications (EuCNC)*, 2017.

[26] ——, “Bit Index Explicit Replication (BIER) Multicasting in Transport Networks,” in *International Conference on Optical Network Design and Modeling (ONDM)*, 2017.

[27] D. Merling *et al.*, “P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast,” *Journal of Network and Computer Applications (JNCA)*, vol. 169, 2020.

[28] ——, “Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4,” *IEEE Access*, vol. 9, 2021.

[29] H. Chen, M. McBride, S. Lindner, M. Menth, A. Wang, G. Mishra, Y. Liu, Y. Fan, L. Liu, and X. Liu, *BIER Fast ReRoute*, https://tools.ietf.org/html/draft-ietf-bier-frr, Jul. 2022.

[30] Y. Desmouceaux *et al.*, “Reliable Multicast with B.I.E.R.,” *Journal of Communications and Networks*, vol. 20, 2018.

[31] ——, “Reliable BIER with Peer Caching,” *IEEE Transactions on Network and Service Management (TNSM)*, vol. 16, 2019.

[32] T. Eckert *et al.*, *Tree Engineering for Bit Index Explicit Replication (BIER-TE)*, https://datatracker.ietf.org/doc/html/draft-ietf-bier-te-arch, Jul. 2021.

[33] W. Braun *et al.*, “Performance Comparison of Resilience Mechanisms for Stateless Multicast using BIER,” in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017.

[34] G. Karypis *et al.*, “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs,” *SIAM Journal on Scientific Computing (SISC)*, vol. 20, 1998.

[35] R. Diekmann *et al.*, "Shape-Optimized Mesh Partitioning and Load Balancing for Parallel Adaptive FEM," *Parallel Computing*, vol. 26, 2000.

[36] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, 1982.

[37] A. Medina *et al.*, "BRITE: An Approach to Universal Topology Generation," in *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001.

[38] B. M. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 6, 1988.

[39] F. Liu *et al.*, "The packet size distribution patterns of the typical internet applications," in *IEEE International Conference on Network Infrastructure and Digital Content*, 2012, pp. 325–332.

1.8 Learning Multicast Patterns for Efficient BIER Forwarding with P4

Learning Multicast Patterns for Efficient BIER Forwarding with P4

Steffen Lindner, Daniel Merling, Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
{steffen.lindner, daniel.merling, menth}@uni-tuebingen.de

***Abstract*—**
Bit Index Explicit Replication (BIER) is an efficient domain-based transport mechanism for IP multicast (IPMC) that indicates receivers of a packet through a bitstring in the packet header. Recently, BIER forwarding has been implemented on 100 Gbit/s hardware using the P4 programming language. However, the implementation requires packet recirculation to iteratively serve one next-hop after another. The objective of this paper is to reduce this inefficiency.

Static multicast groups can be configured on P4 switches so that traffic can be sent to all next-hops without recirculation. We leverage that feature to make BIER forwarding more efficient. However, only a limited number of static multicast groups can be configured on a switch, which is not sufficient to cover all potential port patterns. In a first step, we develop efficient BIER forwarding that utilizes static multicast groups derived from so-called configured port clusters. In a second step, we design port clustering algorithms that observe multicast patterns and compute configured port clusters which are more efficient than randomly selected port clusters. These methods are based on Spectral Clustering, an unsupervised machine learning technique. We perform simulations that underline the effectiveness of this approach to reduce inefficient packet recirculations. We further implement the new forwarding behaviour on programmable hardware and provide a controller that samples BIER packets on the switch, runs the port clustering algorithms, and updates the configured static multicast groups. We validate this open source implementation in a testbed and show that the experimental results are in line with the simulation results.

***Index Terms*—Software-Defined Networking, Bit Index Explicit Replication, Multicast, Resilience, Scalability, Unsupervised Machine Learning**

I. Introduction

IP multicast (IPMC) is an efficient way to distribute one-to-many traffic. It is organized into multicast groups that are identified by unique IP addresses. Traffic of a multicast group is sent to all subscribers along a distribution tree, i.e., nodes replicate and forward packets to specific neighbors towards the subscribers. Therefore, only one packet is sent over each involved link, which reduces the load in comparison to unicast. To that end, core nodes store for each multicast group the neighbours that should receive a packet copy. As a result, traditional IPMC has two scalability issues. First, whenever the composition of an IPMC group changes, signaling to core nodes is necessary to update the neighbors that should receive packet copies. Second, link or node failures, and topology changes may affect multiple multicast groups, which puts high signaling and processing load on core devices.

The IETF proposed Bit Index Explicit Replication (BIER) [1] as an efficient and stateless domain-specific transport mechanism for IPMC traffic. Ingress routers equip an IPMC packet with a bitstring in the BIER header which contains all destinations of the packet within the domain. Core nodes replicate and forward the BIER packet according to its bitstring and the paths from the interior gateway protocol (IGP) which is called routing underlay. Egress routers remove the BIER header, and IPMC processing continues. With BIER, only ingress and egress routers of a domain know IPMC groups and are involved in signalling, but not the core routers.

Recently, we presented an open source BIER implementation for 100 Gbit/s in P4 for the Tofino ASIC hardware [2]. This implementation is inefficient as it requires one processing cycle per next-hop of a BIER packet as packets are transmitted iteratively instead of simultaneously. On the one hand this is due to the fact that packet replication to a dynamic set of outgoing ports is not supported on the specific hardware device. On the other hand, it is difficult to derive the set of outgoing ports from the bitstring within a single processing cycle, which is a general challenge for all switch architectures.

In this paper we present an efficient BIER implementation in P4 for the Tofino ASIC. First, we propose a forwarding algorithm that utilizes static multicast groups to simultaneously forward BIER packets to many outgoing ports. However, the number of configurable static multicast groups is limited and does not suffice to cover all port combinations on a 32-port switch. Therefore, the algorithm leverages smaller "configured" port clusters for which all port combinations are configured as static multicast groups. This allows efficient BIER forwarding within a very few processing cycles (at most 3 or 4 on the Tofino). To further improve the efficiency, we suggest to choose configured port clusters such that they contain ports over which BIER packets are frequently forwarded together. To that end, we propose port clustering algorithms that learn port patterns from sampled BIER traffic and compute configured port clusters that reduce the number of required forwarding cycles. The methods are based on Spectral Clustering which is an unsupervised machine-learning technique. In practice, a controller applies port clustering from time to time on recently sampled BIER traffic and updates the configured port clusters on the switch.

The paper is structured as follows. In Section II and III we describe related work and give an introduction to Bit Index Explicit Replication (BIER). Sections IV and V give

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

a primer on the programming language P4 and cover important aspects of the existing P4-based BIER implementation. Section VI proposes the efficient BIER forwarding algorithm and shows simulation results for arbitrarily selected configured port clusters. Section VII suggests various port clustering methods. Their performance is compared by simulation in Section VIII and by hardware experiments in Section IX. Finally, we conclude the paper in Section X.

II. Related Work

In this section we first review related work on traditional multicast and resilience. Then, we present work related to both SDN- and BIER-based multicast. Finally, we review clustering approaches.

A. Traditional Multicast

Islam et al. [3] and Al-Saeed et al. [4] investigate related work for traditional multicast. The majority of cited papers aim to improve the scalability of traditional IPMC. They present intelligent tree-building mechanisms for multicast to make it more efficient, e.g., by reducing required state, or signaling.

Elmo [5] encodes topology information of data centers in packet headers to improve the scalability of IPMC. It leverages characteristic properties of those topologies to reduce the size of the forwaring information base (FIB) of core routers. The Avalanche Routing Algorithm (AvRA) [6] follows a similar approach where it optimizes link utilization for multicast by leveraging topology characteristics of data center networks. Dual-Structure Multicast (DuSM) [7] separates forwarding structures for high-bandwidth and low-bandwidth traffic to improve scalability and link utilization in data centers. Li et al. [8] optimize the FIB to improve the scalability of traditional multicast in data center networks. To that end, they propose to partition the multicast address space and aggregate those at bootleneck switches.

Application layer multicast (ALM) [9] monitors the traffic on application-specific distribution trees to optimize their structures for the corresponding group objective. Mokhtarian et al. [10] construct minimum-delay trees to reduce latency for delay sensitive data with different requirements like min-average, min-maximum, real-time requirements, etc. Adaptive SDN-based SVC multicast (ASCast) [11] follows a similar approach. The authors describe an integer-linear program to build optimal distribution trees and fast forwarding tables to optimize multicast forwarding in terms of latency and delay for live streaming.

Kaafar et al. [12] present a building scheme for efficient overlay multicast trees based on location-information of subscribers. Boivie et al. [13] propose small group multicast (SGM) which aims at avoiding management and set up overhead for multicast groups with a small number of receivers. To that end, the multicast packets of such groups carry the distribution information in their headers, which avoids signaling in the core. Simple explicit multicast (SEM) [14] stores multicast information only on branching nodes of the distribution tree. Non-branching nodes forward packets to the next-branching node via unicast. Jia et. al. [15] leverage prime numbers and the Chinese remainder theorem to efficiently organize the FIB. They reduce the size of the FIB in core devices and facilitate implementation.

Steiner trees [16] are tree structures that are used to build efficient multicast trees. Many research papers modify Steiner trees to build multicast trees optimized with regard to a specific metric, e.g., link costs [17], delay [18], number of hops [19], number of branch nodes [20], retransmission efficiency [21], or optimal placement of IPMC sources [22].

B. Resilience for Multicast

Shen et al. [23] extend Steiner trees so that distribution trees contain recovery nodes. Such nodes cache multicast traffic for retransmission to cut of receivers after recomputation of the FIB. The authors of [24] investigate resilience of several multicast algorithms against node failures. Kotani et al. [25] deploy primary and backup multicast trees that are identified by a field in the packet header. After failure detection, the source sends its packet over a working backup tree by indicating the backup path in the packet header. Pfeiffenberger et al. [26] propose that each node in a distribution tree is also the root of a backup tree that reaches all downstream destinations over paths that do not include the failed link/node. Nodes switch packets on a backup tree by setting a VLAN tag in the packet header.

C. SDN-Based Multicast

Rückert et al. [27], [28] propose and extend Software-Defined Multicast (SDM) which is an OpenFlow-based multicast platform to facilitate management. It focuses on overlay-based live streaming services for P2P video live streaming. The authors of [29] describe address translation in OpenFlow switches to reduce the number of multicast-dependent forwarding entries in near-to-leaf nodes. To that end, the forwarding action from the last hop towards the receivers is done with a unicast address. Lin et al. [30] implement shared multicast trees between different IPMC groups on OpenFlow switches. Thereby, the number of forwarding entries is reduced. The authors of [31] leverage bloom filters to reduce the number of TCAM-entries that is required for SDN-based multicast.

D. BIER Multicast

In [32], [33] we presented an early prototype of a BIER implementation in P4 for the software switch bmv2 [34]. However, bmv2 yields only low throughput (900 Mbit/s) [35]. Therefore, we developed a P4 implementation of BIER and BIER-FRR for the P4-programmable switching ASIC Tofino [2] with a switching capacity of 3.2 Tb/s, i.e., 100 Gbit/s per port in a 32-port switch. We demonstrated its technical feasibility and performance limits.

Giorgetti et al. [36], [37] presented an OpenFlow implementation of BIER. However, it requires extensive state or controller interaction for efficient BIER forwarding. Furthermore, it is capable of addressing only 20 receivers per packet due do the limited size of MPLS labels which are used to implement arbitrary header fields.

Desmouceaux et al. [38] investigate the retransmission efficiency of BIER. That is, when subscribers signal missing

packets, BIER allows to retransmit packets to only specific subscribers while still forwarding only one packet copy per link. Traditional multicast retransmits either via unicast or to the entire multicast group. The evaluations show that BIER is significantly more efficient than traditional multicast, i.e., it causes fewer retransmitted packets and achieves better link utilization.

BIER with tree engineering (BIER-TE) [39] encodes the entire distribution tree in the packet header to have more control of the paths. Carrier grade minimalist multicast (CGM2) [40] is a novel derivate of BIER-TE. It encodes the distribution tree in a recursive manner in the packet header. Thereby, it can scale to larger networks than BIER-TE. However, CGM2 has not been implemented, yet, and is still under development.

Braun et al. [41] propose 1+1 protection for BIER where traffic is transported on two disjoint trees. As a result, traffic is delivered successfully to receivers even when a failure interrupts one tree.

E. Clustering

Clustering is an unsupervised machine learning technique that solves the problem of identifying clusters of data points in a multidimensional space. Given a set of D-dimensional points $\{x_1, ..., x_N\}$, the goal of clustering is to partition the data into groups/clusters such that points in the same cluster are similar and points in different groups are dissimilar.

k-Means [42] is one of the most applied clustering algorithms. Its incentive is to find an assignment of data points to k cluster centers such that the sum of the squares of the distances of each data point to its cluster center is minimized.

DBSCAN [43] is a density-based clustering algorithm that can form arbitrary clusters and is especially suited for outlier detection. In contrast to k-Means, it is not suited for high-dimensional data sets.

Spectral Clustering [44] is a clustering algorithm that is based on graph properties. It uses the normalized Laplacian of the similarity matrix of the data points to build k clusters. Data points are embedded in $\mathbb{R}^k$ through the so-called spectral embedding. Thereby, the first k eigenvectors of the Laplacian are computed and used to project the data points. Finally, the embedded data points are clustered with a simple clustering algorithm, e.g, k-Means.

III. Bit Index Explicit Replication (BIER)

In this section we give a short primer on BIER. BIER is a domain-based transport mechanism for multicast traffic. It can be explained with three layers as shown in Figure 1. On the IPMC layer sources and receivers send and receive IPMC packets. The BIER layer is responsible for the transport of the IPMC packets from the IPMC sources to the IPMC receivers along paths from the unicast routing, i.e., the routing underlay, through the so-called BIER domain.

The BIER domain consists of three types of BIER devices. First, bit forwarding ingress routers (BFIRs) are the entry points to the BIER domain. They encapsulate IPMC packets with a BIER header for forwarding within the BIER domain. The BIER header contains a bit string that indicates all destinations of the BIER packet. That is, each bit position corresponds to a specific destination. A bit is activated if the corresponding destination should receive a copy of the packet. Second, bit forwarding routers (BFRs) forward BIER packets towards their destinations according to the activated bits in the BIER header. That is, a BFR sends a packet copy to the first next-hop over which at least one destination is reached. It leaves only those bits activated in the bit string of the packet copy which correspond to destinations that are reached via that next-hop, and clears all other bits to prevent duplicates at the receivers. The BFR repeats this procedure until all destinations are served. As a result, the forwarding path of a BIER packet is a tree whose links carry only a single packet copy. Third, bit forwarding egress routers (BFERs) remove the BIER header and pass the IPMC packet to the IPMC layer.

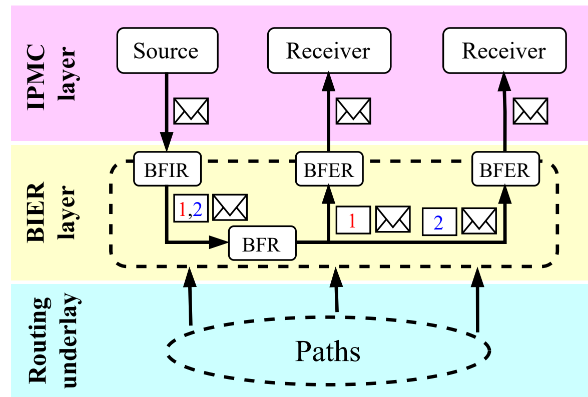


Figure 1: Layered BIER architecture according to [33].

Next-hops on the BIER distribution tree may not be reachable due to link or node failures. In this case, downstream destination nodes do not receive any BIER traffic until BIER forwarding tables are updated Therefore, two BIER fast reroute (BIER-FRR) concepts have been proposed [45] to forward BIER traffic over backup paths from the detection of the failure until BIER forwarding tables have been updated. The methods have been compared in [46] and tunnel-based BIER-FRR has been implemented in [2].

IV. Introduction to P4

In this section we give an overview of P4, explain the P4 processing pipeline, packet cloning, packet recirculation, and multicast groups in P4.

A. P4 Overview

P4 (programming protocol-independent packet processors) [47] is a high-level programming language to describe the data plane of P4-programmable devices. It is applied in a wide range of applications and research [48]. Target-specific compilers map the P4 programs to the programmable processing pipeline of the target devices which are also called targets. The compiled P4 programs offer a control plane API for configuration, e.g., writing forwarding entries, during runtime.

B. P4 Pipeline

Figure 2 shows the abstract pipeline model of P4 [47]. A programmable parser deserializes the packet header and

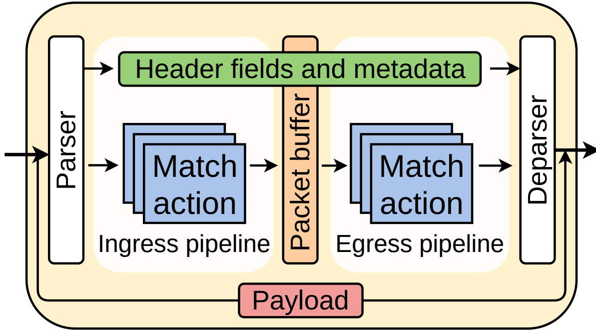


Figure 2: P4 abstract forwarding model according to [47].

stores the information in so-called header fields. The header fields are carried through the pipeline together with packet-specific metadata fields which are comparable to variables from other high-level programming languages. Only header fields and metadata are processed afterwards in the ingress pipeline, i.e., the payload of the packet remains untouched. The ingress pipeline consists of one or more match-action-tables (MATs) that map header fields or metadata to actions. Examples for actions are changing header fields or metadata, or setting the egress port of the packet. After processing in the ingress pipeline, the packet is temporarily buffered so that it can be processed by the egress pipeline which works similarly to the ingress pipeline. Finally, the deparser serializes the possibly changed header fields, forwards the packet through the designated egress port, and discards the metadata.

C. Packet Cloning

P4 has an operation for packet cloning. It sets a flag that the packet should be cloned after its processing in the ingress pipeline has concluded. However, the header fields and metadata of the clone resemble the packet that is initially parsed before the ingress pipeline. Figure 3 shows the concept. After the ingress pipeline has finished, the packet is cloned

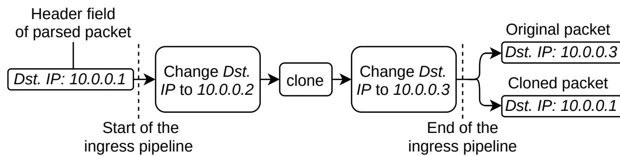


Figure 3: When a packet is cloned, the copy is created only after the ingress pipeline and its header fields are reset to the initial value after the packet has been parsed.

and both the original packet, i.e., with header changes, and the packet copy, i.e., without header changes, enter the egress pipeline where they are processed independently of each other.

D. Packet Recirculation

Packet recirculation in P4 allows a packet to be processed a second time by the entire pipeline, i.e., by ingress and egress pipeline. To recirculate a packet, its egress port, i.e., a special metadata field, is set to a particular port ID that corresponds to a switch-intern recirculation port. The recirculation port functions as a regular port of the switch with the exception that it has no physical connector, i.e., only the switch itself can send to and receive traffic from the recirculation port.

After the packet has been processed by both the ingress and egress pipeline, it is sent to the recirculation port. Afterwards, the packet is processed again as if it has been received on a physical port.

E. Static Multicast Groups

P4 allows controllers to configure multicast groups on forwarding devices. A multicast group consists of a tuple of multicast group identifiers and a set of egress ports. In addition, there is a special metadata field that allows the ingress pipeline to assign a multicast group identifier to a packet. After the ingress pipeline has completed, the packet is replicated to the pre-defined set of egress ports.

In the following we refer to those configured multicast groups as “static multicast groups” to differentiate them from multicast groups of IPMC. A static multicast group is a local mechanism on a switch to simultaneously forward a packet to multiple egress ports.

V. Simple P4-Based BIER Implementation

In this section we review the simple P4-based BIER implementation of [2]. The target is the Intel Tofino high-speed switching ASIC [49]. It is used for a prototype on the Edgecore Wedge 100BF-32X [50] with 32 100 Gbit/s ports. The implementation makes heavy use of packet recirculation, which causes capacity issue. The efficient BIER implementation in Section VI builds upon the simple implementation and greatly reduces the need for recirculations.

A. BIER Processing

BFRs leverage the Bit Index Forwarding Table (BIFT) to determine the next-hops of a BIER packet. We implement the BIFT as common match-action table in P4. For each BFER there is one entry in the BIFT. The match key is a bitstring with only the single bit activated for the corresponding BFER. The other entry fields are a next-hop and a forwarding bitmask (FBM). The FBM is a bit string similar to the BIER bitstring and it indicates the BFERs with the same next-hop. When a packet arrives, the BFR first copies the bitstring of the packet to a temporary metadata field which we call “remaining bits”. The remaining bits indicate the BFERs that still have to be served. Then, the least-significant activated bit in the remaining bits is matched against the BIFT. The match-action table entry returns the corresponding next-hop and FBM for that BFER. The BFR clears all bits in the bitstring of the packet that are not activated in the FBM. Thus, only the bits of BFERs that are reached through this next-hop remain in the BIER bitstring. The BFER further clears all bits in the remaining bits that are activated in the FBM as they have already been served. Afterwards, the clone operation is

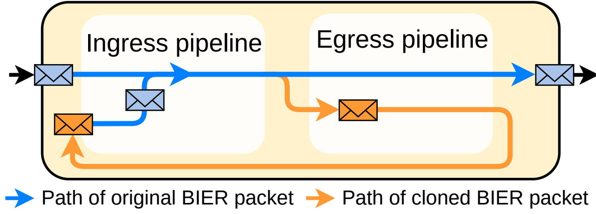


Figure 4: The original BIER packet is sent through an egress port while the packet copy is recirculated.

applied. Figure 4 shows the processing flow of the original and cloned BIER packet. The original packet is sent through the appropriate egress port to reach the selected next-hop. The packet copy is cloned to the egress pipeline and recirculated to a recirculation port. Within the egress pipeline, the BIER bitstring of the packet copy is set to the remaining bits so that only the remaining BFERs are served in the next pipeline iteration.

B. Recirculation Capacity and Problem Statement

The Tofino ASIC has a switch-intern recirculation port which has the same packet processing capacity as regular ports. If its capacity does not suffice, packet loss occurs. To increase the recirculation capacity, physical ports may be turned into loopback mode, and recirculation traffic may be distributed over the internal ports and the loopback ports in a round-robin manner [2]. As these ports cannot be utilized for other traffic, recirculations are costly.

The simple BIER implementation requires $n-1$ recirculations for BIER packets with n next-hops. This approach obviously does not scale well with increasing number of next-hops and traffic rate. The objective of this paper is a more efficient P4-based implementation that requires fewer recirculations per BIER packet (see Section VI) and an optimized configuration thereof using clustering methods (see Section VII).

VI. Efficient BIER Forwarding with P4

We explain how static multicast groups can be leveraged to make BIER forwarding using P4 more efficient, and how BIER-FRR can be integrated. To demonstrate the efficiency of the new forwarding algorithm, we present a simulative performance study.

A. Efficient BIER Forwarding with Static Multicast Groups

We first explain how BIER forwarding can profit from configured port clusters consisting of static multicast groups such that multiple next-hops can be served within a single processing cycle. Then we explain how the forwarding algorithm determines a port cluster and the appropriate static multicast group for a BIER packet, and forwards it.

The presented algorithm is specific to P4 and the architecture of the Tofino ASIC. However, efficient forwarding algorithms for any switch architecture need to determine the set of egress ports for a BIER packet. This is a difficult task as bitstrings are at least 256 bits large. Therefore, the presented approach may be a base for efficient BIER forwarding on other switch architectures.

1) Use of Static Multicast Groups: The idea to make the BIER forwarding more efficient is the use of static multicast groups so that multiple egress port can be simultaneously served.

A naive solution is configuring static multicast for all possible combinations of egress ports. When a packet arrives, the set of egress ports is determined and the corresponding static multicast group forwards the packet to all needed egress ports without packet recirculation. However, on a 32 port switch this requires $2^{32} = 4294967296$ static multicast groups, which exceeds the number of configurable static multicast ports.

We propose now a more sophisticated approach which requires fewer static multicast groups. We define so-called "configured port clusters" (or port sets) $\mathcal{C} = \{C_1, ..., C_k\}$ such that they cover together all ports of a switch. For each port set C_i, static multicast groups $M_j \subseteq C_i$ are configured for all subset of ports in C_i. Thus, a configured port cluster C implies

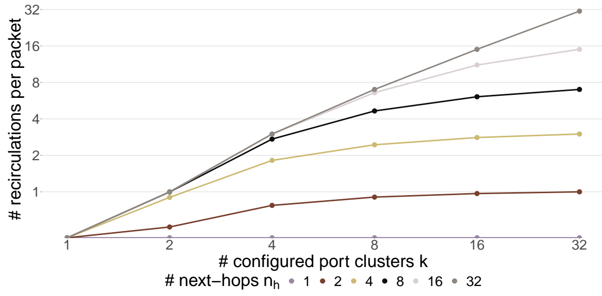
$$m(C) = 2^{|C|} - |C| - 1 \quad (1)$$

static multicast groups that need explicit configuration on the switch; the empty group and groups with only a single destination do not need to be configured. On a 32-port switch three port clusters with 10, 11, and 11 ports may be configured, which requires in total 5085 explicitly configured static multicast groups. This is well feasible on a switch like the Tofino which supports up to $2^{16} = 65536$ static multicast groups[1]. Moreover, the administrator may set another threshold m_{max} on the number of static multicast groups usable for efficient BIER forwarding. With this approach, a BIER packet needs to be sent to at most $|\mathcal{C}|$ static multicast groups, which requires $|\mathcal{C}| - 1$ recirculations instead of $n_h - 1$ with n_h being the number of next-hops of a BIER packet.

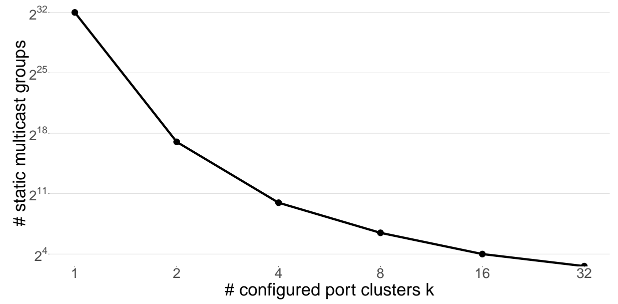
2) Forwarding Procedure: We first describe how the forwarding procedure selects a set of configured port cluster $\mathcal{S}_i \subseteq \mathcal{C}$ for BIER forwarding, and then we present how the appropriate static multicast group is chosen from a selected configured port cluster $C_j \in \mathcal{S}_i$.

a) Selection of Set of Configured Port Clusters: We consider all subets of configured port clusters $\mathcal{S}_i \subseteq \mathcal{C}$. C-FBM($\mathcal{S}_i$) is the combined forwarding bitmask of such a subset. We set up a match-action table with one entry per subset $\mathcal{S}_i$ in increasing order with regard to subset size $|\mathcal{S}_i|$. The entry is the C-FBM($\mathcal{S}_i$). The objective is to find the smallest subset that serves all BFERs of a BIER packet. To that end, the bitstring of a packet is bitwise ANDed with the complement of the C-FBM in the match-action table. We define a match if the result of that operation is zero. Then, all BFERs of the BIER packet are served by the corresponding subset $\mathcal{S}_i$. Due to the order within the match-action table, the first match $\mathcal{S}_i$ is the smallest subset with that property. The first configured port cluster C_j in that subset $\mathcal{S}_i$ is selected for the remainder of the forwarding process.

[1]The actual usable number of available resources depends on the program complexity.



(a) Average number of recirculations per packet for $n_h \in \{1, 2, 4, 8, 16, 32\}$ random next-hops.



(b) Number of required static multicast groups.

Figure 5: Average number of recirculations and number of static multicast groups for $k \in \{1, 2, 4, 8, 16\}$ configured port clusters.

We consider a trivial example with two BFERs reachable over Port 1 and Port 2. Configured port clusters are $\mathcal{C} = \{C_1 = \{1\}, C_2 = \{2\}\}$; the C-FBMs for all subsets $\mathcal{S}_i \subseteq \mathcal{C}$: C-FBM($\emptyset$) = 00, C-FBM($\{C_1\}$) = 10, C-FBM($\{C_2\}$) = 01, C-FBM($\{C_1, C_2\}$) = 11. We assume the bitstring of a BIER packet to be 11; then only $\mathcal{S}_i = \{C_1, C_2\}$ can cover all BFERs of the packet.

b) Selection of the Static Multicast Group: We now determine the appropriate static multicast group from the configured port cluster C_j. To that end, we take a similar approach as in Section VI-A2a. We set up a match-action table for C_j which has an entry for any static multicast group $M_h \subseteq C_j$. The entries are sorted by increasing group size $|M_h|$ and contain the C-FBM of the corresponding multicast group. Single ports are also considered as static multicast groups although they do not require explicit configuration on the switch. The bitstring of a BIER packet is first ANDed with the C-FBM of the selected configured port cluster C_j. This excludes all BFERs from the bitstring that cannot be served by C_j. The result is bitwise ANDed with the complement of the C-FBM(M_h) of the multicast groups in the table entries. We define a match if the result is zero. Due to the increasing order of entries in the match-action table, the first match refers to the smallest static multicast group M_h within the configured port cluster that covers all relevant BFERs.

c) Forwarding and Bitstring Adaptation: At the end of the ingress pipeline, the activated bits in C-FBM($\mathcal{S}_i$) are deactivated in the bitstring of the orginal packet; if the bitstring is not zero, the packet is recirculated. In addition, a clone of the packet is sent to all egress ports of the selected static multicast group M_h. The egress pipelines of these ports clear all bits in the packet's bitstring that are not activated in the FBM of the corresponding port and then they transmit the packets.

B. Integration of BIER-FRR

The proposed efficient forwarding scheme is compatible with BIER-FRR if BIER-FRR is integrated as follows. First, the switch processes the egress ports that are affected by a failure, i.e., a failed link or a failed node. To that end, the BIER packets are forwarded by regular BIER forwarding but over alternate ports. When all affected egress ports have been served, the BIER packet is recirculated and the remaining ports, i.e., working ports, are processed by the presented, efficient forwarding algorithm. This approach prevents duplicates at subscribers and unnecessary double transmissions of the same packet over one link. Details are given in [2].

C. Simulative Performance Evaluation

We evaluate the concept of static multicast groups for efficient BIER forwarding through the following experiment. We examine different numbers of disjoint configured port clusters $k \in \{1, 2, 4, 8, 16, 32\}$. With k configured port clusters and a 32 port switch, each configured port cluster contains $\frac{32}{k}$ ports. Further, we simulate BIER packets with $n_h \in \{1, 2, 4, 8, 16, 32\}$ random next-hops. They are processed by the different configured port clusters. Figure 5(a) and Figure 5(b) show the average number of recirculations per packet and the required static multicast groups.

The average number of recirculations increases with the number of next-hops n_h and the number of configured port clusters k. In fact, the number of recirculations is bound by $k-1$. For $k = 32$, the results are equivalent to the simple BIER forwarding. Higher values of k lead to smaller configured port clusters, and hence, to fewer next-hops that can be served in one shot. The number of required static multicast groups decreases with the number of configured port clusters k. To keep the number of recirculations low, larger configured port clusters should be preferred. However, the number of available static multicast groups may be limited due to technical reasons or based on administrative decisions.

In the given traffic model, we randomly selected next-hops for BIER packets. This is not a realistic model for multicast traffic. The next-hops of subsequent BIER packets are likely to be correlated and so are the ports over which the packets are sent. Therefore, some configured port clusters reduce the average recirculation more than others. To effectively minimize the number of recirculations, it is necessary to form meaningful configured port clusters that take the current traffic model into account.

VII. Port Clustering Algorithms for Efficient BIER Forwarding

In this section, we first illustrate the optimization potential of efficient BIER forwarding through configuration of appropriate port clusters. Then, we present three clustering algorithms to reduce the average recirculations per packet: random port clustering (RPC) as a simple baseline, port clustering based on Spectral Clustering (PCSC), and recursive clustering with overlaps (RPCO) which also leverages Spectral Clustering for subroutines. For the latter two algorithms we present a graph embedding method that turns ports of sampled packets into a graph structure from which the algorithms learn correlated port clusters.

A. Optimization Potential and Approach

The bits in the BIER header require a packet to be sent to a certain set of next-hops, and, thereby, to specific ports of a switch. To be brief, we talk about "ports of a packet". In the previous section we showed how multiple ports of a BIER packet can be served at once to speed up the forwarding process. For example, port clusters $\{1, .., 8\}$, $\{9, .., 16\}$, $\{17, .., 24\}$, and $\{25, .., 32\}$ may be configured. Then, a BIER packet needs to be processed at most four times, i.e., it must be recirculated three times, no matter how many BFERs are set in the BIER header. If a packet has only ports in the range $\{1, .., 8\}$, the packet does not need to be recirculated at all. However, if a packet has ports $\{1, 9, 17, 25\}$, it still requires three recirculations.

We now assume that ports of a packet are not random but correlated. That is, certain ports sets tend to occur together. We call them correlated port clusters. We propose to learn these correlated port clusters from sampled traffic and to utilize them as configured port clusters. Then it is likely that BIER packets can be forwarded with fewer processing steps and, thereby, the number of recirculations may be reduced. In practice, a controller can continuously sample multicast traffic from a switch, learn the correlated port clusters of the sampled multicast traffic, and adjust the configured port clusters on the switch.

Large configured port clusters require lots of static multicast groups, but they have the potential to effectively reduce the number of recirculations. A constraint is the maximum number m_{max} of static multicast groups usable for configured port clusters which may be a technical limit or defined by the administrator.

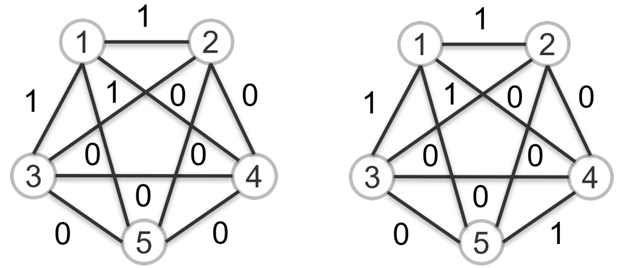
B. Random Port Clustering (RPC)

With RPC, n_p ports are randomly partitioned into approximately k equal-size clusters. The number of clusters k is determined such that the resulting number of configured static multicast groups is at most m_{max}. As the algorithm is trivial, we do not provide any further details. The method will serve as a baseline for a performance comparison.

C. Port Clustering based on Spectral Clustering (PCSC)

We first present a graph embedding method that turns ports of sampled packets into a graph structure from which the algorithms learn correlated port clusters. Then we present the PCSC algorithm which is based on Spectral Clustering. It partitions n_p ports into approximately equal-size port clusters.

1) Graph Embedding: We embed the port information of sampled packets into a graph which is needed by the algorithms for PCSC and RPCO. The nodes of the graph represent the ports of a switch. The graph is fully connected and the edges have weights. All weights are initially zero. The embedding iteratively processes the sampled packets. For any two ports of a packet, the weight of the link between these ports is increased by one. Figure 6(a)-Figure 6(b) illustrate how two sampled packets with ports $\{1, 2, 3\}$ and $\{4, 5\}$, respectively, modify an embedded graph with 5 nodes whose edges are initially all zero.



(a) The edge weights between egress ports 1, 2, and 3 are increased by one.

(b) The edge weights between egress ports 4 and 5 are increased by one.

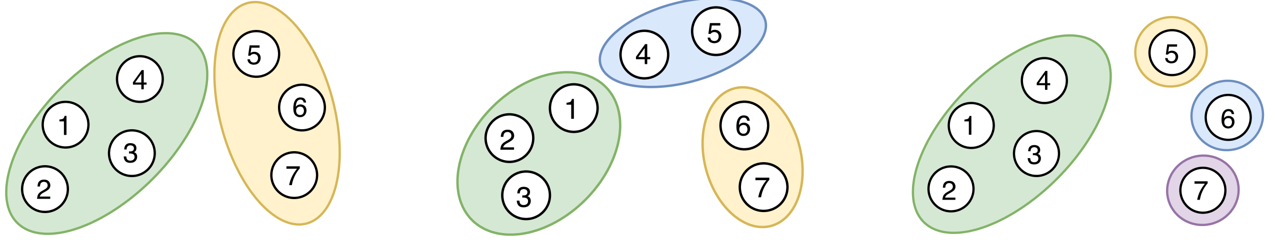
Figure 6: Graph embedding: a full-mesh graph is augmented by port information from sampled packets: high edge weights indicate port pairs that frequently occur together in a BIER packet.

2) PCSC Algorithm: We first develop a metric for port clusters that correlates with the number of recirculations needed for the sampled traffic. Then we propose pseudocode for PCSC that minimizes that number while respecting the number of usable static multicast groups.

a) Metric: We consider two port clusters C_1 and C_2. The clustering is good if only a few BIER packets need to be sent through ports of C_1 and C_2. We identify a metric for the graph embedding that correlates with that number of packets although it is not an exact measure for it. The function $cut(C_1, C_2)$ is the sum of the weights on the edges between any two nodes $v_1 \in C_1$ and $v_2 \in C_2$. It gives an upper bound on the number of packets with ports in both C_1 and C_2. It is an upper bound and not the exact number as a packet may have multiple ports from C_1 and/or C_2. To assess whether the clustering is good, we need to relate $cut(C_1, C_2)$ to the overall number of nodes in the considered clusters. This can be done with the so-called normalized cut (Ncut) and is given below, generalized for multiple clusters.

$$Ncut(C_1, ..., C_k) = \sum_{i=1}^{k} \frac{cut(C_i, \overline{C_i})}{vol(C_i)}$$

Thereby, $cut(C_i, \overline{C_i})$ measures the sum of the edge weights between nodes in C_i and nodes that are not in C_i ($\overline{C_i}$). The



(a) Two almost equal-size port clusters need 15 static multicast groups.

(b) Three almost equal-size port clusters need 6 static multicast groups.

(c) The optimal port clusters have unequal size and need 11 static multicast groups.

Figure 7: Most BIER packets are sent to ports $\{1, 2, 3, 4\}$, $\{3, 4\}$, and $\{4, 5\}$ on a 7-port switch and the maximum number of usable static multicast groups is $m_{max} = 12$. PCSC produces equal-size port clusters while the optimum port clusters minimizing the overall number of recirculations has unequal size.

function $vol(C_i)$ sums up the edge weights of all nodes in C_i – as a result, edge weights between nodes within the cluster are counted twice, weights of outgoing edges are counted once. The objective is to find clusters $C_1, ..., C_k$ that minimize the normalized cut. Ncut is known to be NP hard and therefore cannot be solved efficiently. However, Spectral Clustering is a relaxation of Ncut. It yields a partition $\mathcal{C}$ with preferably equal-size clusters $C_i \in \mathcal{C}$ and can be solved efficiently [44].

b) Pseudocode for PCSC: PCSC is described in Algorithm 1. It first performs the graph embedding for the set of sampled packets $\mathcal{S}$ and the given number of nodes n_p. Then, Spectral Clustering is called to provide a partition $\mathcal{C}$ of the n_p nodes into k clusters. This is performed in a loop, starting from a single cluster up to n_p clusters. As soon as a partition $\mathcal{C}$ is found that requires at most m_{max} static multicast groups, the algorithm stops and $\mathcal{C}$ is returned. It is the clustering with the lowest number of clusters that can be configured with m_{max} static multicast groups.

Algorithm 1 PCSC

```
Input: samples: S
       number of ports: n_p
       number of multicast groups: m_max
graph = graphEmbedding(n_p, S)
for k from 1 to n_p do
    C = SpectralClustering(graph, k)
    if number of multicast groups for C ≤ m_max then
        return C
    end
end
```

D. Recursive Port Clustering with Overlap (RPCO)

We first explain two major shortcomings of PCSC. Then we explain how RPCO solves these shortcomings. Finally, we give a high-level pseudocode description of RPCO.

1) Shortcomings of PCSC: PCSC has two major shortcomings. First, if the configured port clusters cannot be built, the number of clusters is increased by one. As a result, an important cluster that significantly reduces the number of recirculations may not be built although a less important cluster could be split to save static multicast groups.

We illustrate that with a 7-port switch and $m_{max} = 12$ usable static multicast groups. We assume that most multicast packets are sent to port clusters $\{1, 2, 3, 4\}$, $\{3, 4\}$, and $\{4, 5\}$. When PCSC is called with $k = 2$, the clusters in Figure 7(a) may be returned which require 15 static multicast groups, which exceeds m_{max} so that it is not a valid solution. Therefore, PCSC increases k to 3, which may return the clusters in Figure 7(b) which require only 6 static multicast group. As this is feasible, this clustering is PCSCs final result. However, the optimal clustering that minimizes the overall number of recirculations might be the one in Figure 7(c) with 4 unequal-size clusters. They require 11 static multicast groups, which is also feasible.

Second, PCSC creates disjoint clusters. This, however, may not be optimal. We illustrate that by a small example. We consider packets with ports $\{1, 2, 3\}$ and $\{2, 3, 4\}$ and $m_{max} = 8$ usable static multicast ports. A single, large cluster $C = \{1, 2, 3, 4\}$ requires $m(C) = 11$ static multicast groups so that it cannot be configured. When working with smaller, non-overlapping clusters, it is not possible to cover the port sets of both packets with only a single port cluster. However, when working with overlapping port clusters $C_1 = \{1, 2, 3\}$ and $C_2 = \{2, 3, 4\}$, only 7 static multicast groups are needed[2], which is feasible. Moreover, the port sets of both packets can be covered. Thus, overlapping clusters may help to further reduce the number of recirculations with a limited number of usable static multicast groups.

2) Design Ideas: We discuss major design ideas of RPCO. If the number of usable static multicast groups m_{max} does not suffice to configure k clusters proposed by Spectral Clustering, RPCO selects the clusters that reduce recirculations in the most efficient way and recursively re-clusters the remaining clusters. To that end, we review and adapt the knapsack algorithm to

[2]When working with overlapping port clusters, the static multicast groups required by multiple port clusters need to be configured only once on the switch.

select the clusters that reduce recirculations most efficiently. Given a clustering, we further suggest how to add nodes also to other clusters they are not yet part of, which facilitates cluster overlaps.

a) The Knapsack Algorithm: In the knapsack problem [51], a set of items is given, and each item has a weight and a value. The knapsack objective is to select items such that their overall weight is less than a given limit while their overall value is maximized.

We apply the knapsack algorithm as follows. The set of items is given as set of port clusters $\mathcal{C} = \{C_1, ..., C_k\}$. The value of a port cluster C_i is given by the number of recirculation it saves for the set of packets $\mathcal{S}$ which is evaluated by simulation. The weight of a port cluster C_i is given by its number of required static multicast groups $m(C_i)$. The limit is the number of usable static multicast groups. The algorithm selects those clusters that maximize the number of saved recirculations with the available static multicast groups.

b) Adding Single Nodes to Multiple Clusters: We first define the so-called port-cluster relevance $r(x, C)$ of a port x and a cluster C, $x \notin C$. Then, we explain how the port-cluster relevance is used to add single nodes to multiple clusters.

The port-cluster relevance measures the connectivity between port x and cluster C. It is the sum of the edge weights w between x and C, i.e., $r(x, C) = \sum_{y \in C} w(x, y)$.

Ports are initially assigned to a cluster with Spectral Clustering. However, ports may also be important for other clusters. The list of all port-cluster pairs sorted by decreasing port-cluster relevance suggests the order in which nodes should be additionally added to another cluster provided the remaining static multicast groups suffice. As a result, a partition of ports becomes a port clustering with overlaps.

3) Pseudocode for RPCO: We give a high-level pseudocode for RPCO and refer to the Github repository[3] for details.

Algorithm 2 describes the outer control loop of RPCO. First, the graph embedding of the samples $\mathcal{S}$ is computed and stored in $graph$. Then, the best clustering $\mathcal{C}_{best}$ is initialized with single node clusters. A graph with n_p nodes (number of ports on the switch) can be partitioned into up to n_p clusters. Therefore, the subsequent loop is called with k between 1 and n_p. Within the loop, the current clustering $\mathcal{C}$ is initialized empty and the number of remaining static multicast groups m_{left} is initialized with m_{max}. Both $\mathcal{C}$ and m_{left} are global variables so that they can be modified by subroutines. RecursiveClustering computes a partition of all nodes and stores it in $\mathcal{C}$. Details of the procedure will be explained later. Then, OverlapClusters utilizes remaining usable static multicast groups m_{left} to add nodes to other clusters they are not yet part of (see Section VII-D2b). This leads to overlapping clusters. Afterwards, the best clustering $\mathcal{C}_{best}$ is updated by $\mathcal{C}$ if $\mathcal{C}$ requires fewer recirculations than the best clustering. The function Recirculations$(\mathcal{C}, \mathcal{S})$ computes the number of recirculations required for clustering $\mathcal{C}$ for the packets in $\mathcal{S}$. Finally, RPCO returns the best clustering of all switch ports that minimizes the number of recirculations for the samples $\mathcal{S}$.

[3] Github: https://github.com/uni-tue-kn/rpco

Algorithm 2 RPCO

Input: samples: $\mathcal{S}$
number of ports: n_p
max. number of multicast groups: m_{max}

$graph$ = GraphEmbedding$(n_p, \mathcal{S})$
$\mathcal{C}_{best} = \{\{1\}, ...\{n_p\}\}$
for $k \in [1, n_p]$ **do**
  $\mathcal{C} = \{\emptyset\}$
  $m_{left} = m_{max}$
  RecursiveClustering$(graph, k)$
  OverlapClusters$(graph)$
  if *Recirculations*$(\mathcal{C}, \mathcal{S})$ <*Recirculations*$(\mathcal{C}_{best}, \mathcal{S})$ **then**
    $\mathcal{C}_{best} = \mathcal{C}$
  end
end
return *Best port clustering* $\mathcal{C}_{best}$

RecursiveClustering is described in Algorithm 3. If the graph contains only a single node v, the node is added as a separate cluster to $\mathcal{C}$ and the recursion ends. Otherwise, Spectral Clustering is executed to produce clustering $\mathcal{C}'$ with the desired number of clusters k. Then, the cluster set $\mathcal{C}^*$ is identified which makes best use of the remaining static multicast groups m_{left} to reduce recirculations. All clusters in $\mathcal{C}^*$ are added to the current clustering result $\mathcal{C}$ and m_{left} is decreased by their number of required static multicast groups. The clusters not selected by knapsack ($\mathcal{C}' \setminus \mathcal{C}^*$) are recursively clustered. To that end, the corresponding embedded subgraph is computed. The recursion ends if either the recursion was called with a single node or if all clusters $\mathcal{C}'$ can be selected.

Algorithm 3 RecursiveClustering

Input: graph embedding: graph
number of clusters: k

if $graph$ contains only the single node v **then**
  $\mathcal{C} = \mathcal{C} \cup \{\{v\}\}$
  return
end
$\mathcal{C}'$ = SpectralClustering$(graph, k)$
$\mathcal{C}^* =$ knapsack$(\mathcal{C}', \mathcal{S}, m_{left})$
for $C \in \mathcal{C}^*$ **do**
  $\mathcal{C} = \mathcal{C} \cup \{C\}$
  $m_{left} = m_{left} - m(C)$
end
for $C \in \mathcal{C}' \setminus \mathcal{C}^*$ **do**
  $subgraph$ = subgraph of $graph$ limited to nodes in C
  RecursiveClustering$(subgraph, 2)$
end

VIII. Simulative Performance Comparison

In this section we compare the performance of the three port clustering methods Random Port Clustering (RPC), Port Clustering based on Spectral Clustering (PCSC), and Recursive Port Clustering with Overlaps (RPCO). We first develop a model for correlated multicast traffic and explain the

performance evaluation methodology. Then, we compare the performance of the mentioned clustering methods for various correlated multicast traffic models. Finally, we compare the runtime of the algorithms.

A. Traffic Model and Evaluation Methodology

We define a simple model for correlated multicast traffic and explain the methodology for the subsequent comparison of the port clustering methods.

1) Model for Correlated Multicast Traffic: In Section VI-C we utilized a model for multicast traffic that assumes random ports for subsequent multicast packets. However, random ports are not realistic for two reasons. First, subsequent multicast packets belong to a set of active multicast groups and packets of a multicast group have identical ports as long as the groups do not change. Second, receivers of multicast groups are users or connected upstream aggregation points in specific time zones, geographical regions, or neighborhoods. Therefore, we assume the users have common interests for certain multicast content so that they belong to multicast groups with correlated receivers. We have not found any literature studying this issue and think this would be useful future work.

We propose a model for correlated multicast traffic for use in the subsequent performance comparison. We define a set of generating port clusters $\mathcal{C}_g = \{C_1, C_2, ..., C_k\}$ from which ports of a packet are preferentially chosen. First, we randomly choose one generating port cluster C_i; thereby all C_i have equal probability. Then, we determine a random number of ports which is equally distributed between 1 and the size $|C_i|$ of the chosen cluster. We draw these ports with a probability p from C_i (without duplicates) and with probability $1-p$ from ports outside C_i (without duplicates).

For $p = 1$, all ports of a sampled BIER packet are from a single, generating port cluster C_i. In that case, if the generating port clusters $\mathcal{C}_g$ are configured for efficient BIER forwarding, BIER packets can be recirculated without recirculation. As p decreases, a sampled BIER packet is likely to have increasingly more ports outside the selected generating port cluster C_i. That means, the resulting multicast traffic is more random and more recirculations are needed. We take p as a measure for port correlation in the generated multicast traffic.

2) Evaluation Methodology: The objective of port clustering algorithms for efficient BIER forwarding is the reduction of recirculations. Therefore, we take the average number of recirculations per packet as performance metric for the subsequent comparisons.

We generate 1000 BIER packets. Based on these packets we compute sets of port clusters for optimized configuration using the considered port clustering methods and various numbers of usable static multicast ports m_{max}. Then, we generate another 10000 packets and simulate efficient BIER forwarding using the optimized configuration. We count the number of recirculations and compute the average number of recirculations per packet. We conduct the experiments 100 times and produce 95% confidence intervals for the average number of recirculations. As they are very small, we omit them in the figures for the sake of readability.

B. Performance Comparison of Port Clustering Methods

We compare the efficiency of the port clustering algorithms for different traffic models. We consider disjoint and overlapping generating port clusters of equal and unequal size. We choose the models such that they all lead to 4.5 ports per BIER packet, which makes their results comparable.

1) Multicast Traffic Generated from Disjoint Port Clusters: We study correlated multicast traffic generated from disjoint generating port clusters. We consider symmetric and asymmetric generating port clusters.

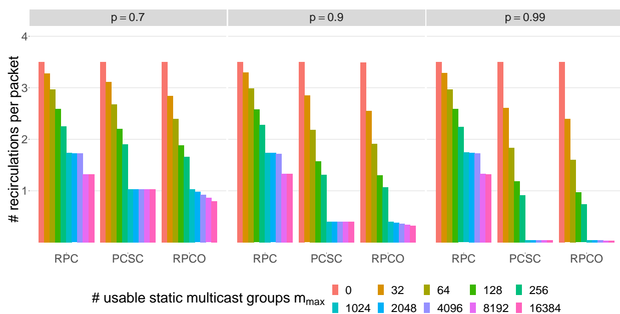
a) Symmetric Generating Port Clusters: We consider four symmetric, disjoint, generating port clusters of size 8: $C_1 = \{1, .., 8\}$, $C_2 = \{9, .., 16\}$, $C_3 = \{17, .., 24\}$, $C_4 = \{25, .., 32\}$. If they are used for configuration, $4 \cdot (2^8 - 8 - 1) = 988$ static multicast groups are needed.

Figure 8(a) shows the average number of recirculations per packet for traffic models with port correlation $p \in \{0.7, 0.9, 0.99\}$, for usable static multicast groups $m_{max} \in \{0, 32, 64, 128, 256, 10.24, 2048, 4096, 8192, 16384\}$, and for the port clustering methods RPC, PCSC, and RPCO.

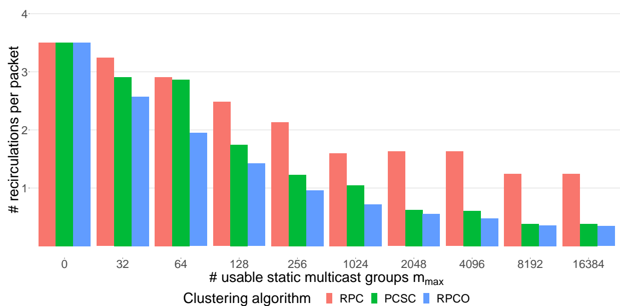
If no static multicast group is available for efficient BIER forwarding ($m_{max} = 0$), the port clustering is disabled, and the forwarding behaviour is the same as the one for simple BIER forwarding. Therefore, packets with 4.5 ports on average require 3.5 recirculations on average. Increasing the number of usable multicast groups m_{max} allows efficient BIER forwarding to decrease the average number of recirculations per packet. This holds for all traffic models and for all port clustering methods. However, if sufficient static multicast groups are available, the degree to which the average number of recirculations can be reduced depends on the port correlation p and the port clustering method.

If a packet with l ports is generated from a specific generating port cluster, all the ports are taken from that cluster with a probability of p^l. Setting $l = 4.5$ yields 20.1% for $p = 0.7$, 62.2% for $p = 0.9$, and 95.6% for $p = 0.99$. Thus, the chosen traffic models are quite divers. For port correlation $p = 0.7$, the average number of recirculations are similar for all considered port clustering algorithms. The advanced port clustering algorithms hardly outperform the random method due to the lack of sufficient port correlation in the generated multicast traffic. For port correlation $p = 0.99$, most packets are entirely drawn from a single generating port cluster. As a result, the advanced packet clustering methods lead to significantly fewer packet recirculations than random clustering. With $m_{max} = 1024$ or more usable multicast groups, PCSC and RPCO reduce the average number of recirculations to almost zero. Apparently they are able to learn the right port clusters. The generating port clusters are optimal for configuration; as mentioned above, they require 988 static multicast groups. This explains why $m_{max} = 512$ or fewer static multicast groups require more recirculations, also with advanced port clustering methods. The results in Figure 8(a) show that PSCS and RPCO lead to about the same number of recirculations per packet for symmetric, disjoint, generating port clusters.

In the following, we choose port correlation $p = 0.9$ as this generates sufficiently correlated multicast traffic with

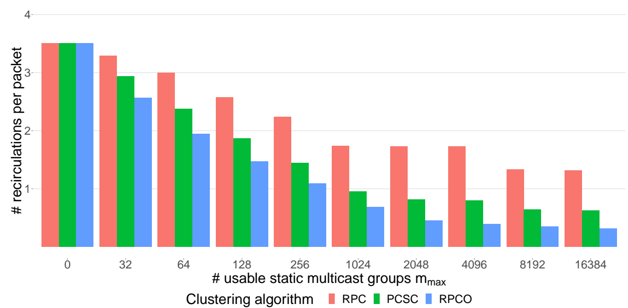


(a) Traffic sampled from four generating port clusters of size 8 with different port correlation p.

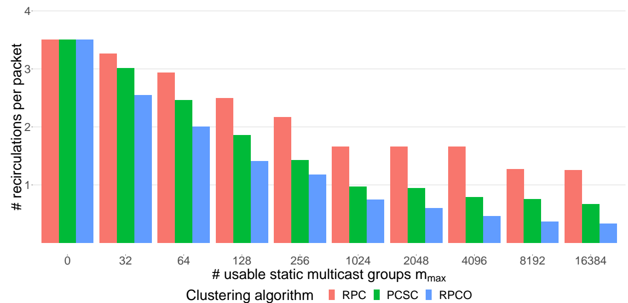


(b) Traffic sampled from four clusters of size 12, 10, 6, 4 with port correlation $p = 0.9$.

Figure 8: Impact of port clustering methods and number m_{max} of usable, static multicast groups on the average number of recirculations per packet; multicast traffic is sampled from disjoint generating port clusters.



(a) Traffic sampled from six clusters of size 8.



(b) Traffic sampled from six clusters of size 12, 10, 8, 8, 6, 4.

Figure 9: Impact of port clustering methods and number m_{max} of usable, static multicast groups on the average number of recirculations per packet; multicast traffic is sampled from overlapping, generating port clusters with port correlation $p = 0.9$.

substantial port deviation from the generating port clusters.

b) Asymmetric Generating Port Clusters: We consider four asymmetric, disjoint, generating port clusters of size 12, 10, 6, 4: $C_1 = \{1, .., 12\}$, $C_2 = \{13, .., 22\}$, $C_3 = \{23, .., 28\}$, and $C_4 = \{29, .., 32\}$. If used for configuration, they require $(2^{12}-12-1)+(2^{10}-10-1)+(2^6-6-1)+(2^4-4-1) = 5164$ static multicast groups.

Figure 8(b) illustrates the average number of recirculations per packet for port correlation $p = 0.9$. Again, more usable static multicast groups cause fewer recirculations. We now observe that RPCO reduces the average number of recirculations to lower numbers than PCSC, in particular for $m_{max} \leq 4096$. For larger m_{max}, PCSC and RPCO lead to almost equal results. This is in line with the design goal of RPCO: it makes better use of a limited number of static multicast groups than PCSC by proposing unequal-size port clusters. For $m_{max} = 64$, PCSC causes 3 recirculations per packet while RPCO causes only 2. For port correlation $p = 0.99$, which is not shown in the figure, both PCSC and RPCO reduce the average number of recirculations to almost zero for $m_{max} \geq 8192$.

2) Multicast Traffic Generated from Overlapping Port Clusters: We study the performance of the presented clustering algorithms for overlapping, generating port clusters.

a) Symmetric Generating Port Clusters: We consider six overlapping, generating port clusters of size 8: $C_1 = \{1, .., 8\}$, $C_2 = \{6, .., 13\}$, $C_3 = \{11, .., 18\}$, $C_4 = \{17, .., 24\}$, $C_5 = \{22, .., 29\}$, and $C_6 = \{28, .., 32, 1, .., 3\}$. Configuring them as port clusters requires $6 \cdot (2^8 - 8 - 1) - 4 \cdot (2^3 - 3 - 1) - 2 \cdot (2^2 - 2 - 1) = 1464$ static multicast groups.

Figure 9(a) indicates the average number of recirculations per packet for port correlation $p = 0.9$. Here, PCSC outperforms RPC, and RPCO outperforms PCSC for any number $m_{max} > 0$ of usable static multicast groups. While PCSC computes only disjoint port clusters, RPCO may yield overlapping port clusters. This can lead to fewer recirculations when frequently observed port groups of packets are partly overlapping. For port correlation $p = 0.99$, which is not shown in the figure, only RPCO reduces the average number of recirculations to almost zero for $m_{max} \geq 2048$.

b) Asymmetric Generating Port Clusters: We consider six overlapping, generating port clusters of size 12, 10, 8, 8, 6, 4: $C_1 = \{1, .., 12\}$, $C_2 = \{9, .., 16\}$, $C_3 = \{16, .., 19\}$, $C_4 = \{18, .., 23\}$, $C_5 = \{22, .., 29\}$, and $C_6 = \{27, .., 32, 1, .., 5\}$. Configuring them as port clusters requires 5630 static multicast groups.

Figure 9(b) illustrates the average number of recirculations per packet for port correlation $p = 0.9$. The results are very similar to those in Figure 9(a), only a few recirculations more are required. That means, PCSC clearly outpeforms RPC, and RPCO outperforms PCSC. For $p = 0.99$ and $m_{max} \geq 8192$, which is not shown here, RPCO even reduces the average number of recirculations to almost zero. That is, it is able to find optimal clusters for configuration even under

challenging conditions (overlapping, unequal-size, generating port clusters).

C. Runtime

The presented clustering algorithms, especially RPCO, seem rather complex at first glance. We measure the runtime of the presented algorithms for the evaluation in Section VIII-B2b. The experiments are executed on a 2022 Mac Studio with M1 Max and 32 GB of RAM. Figure 10 compiles the results.

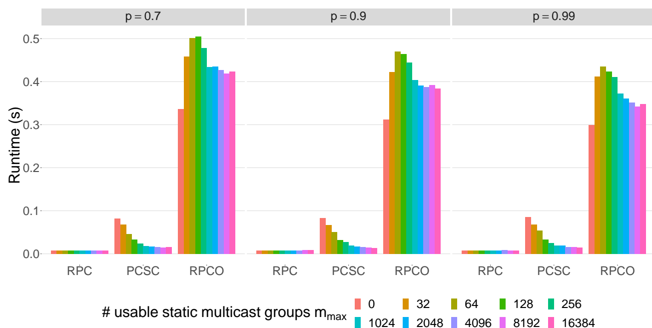


Figure 10: Average runtime in seconds for the clustering algorithms RPC, PCSC, and RPCO while performing experiments for Section VIII-B2b.

Random Port Clustering (RPC) has the shortest runtime with at most 9 ms. It partitions all ports into equal-size clusters and its runtime is therefore independent of port correlation p. PCSC reveals the second lowest runtime with up to 86 ms. It calls the Spectral Clustering subroutine at most n_p times where n_p is the number of ports. PCSC's runtime decreases with increasing m_{max} because larger values of m_{max} lead to fewer subroutine calls (return leaves the loop in Algorithm 1). RPCO has the longest runtime with up to 527 ms. It also performs n_p iteration steps but may call Spectral Clustering multiple times within a single iteration step. Its runtime primarily depends of the number of recursive calls. With decreasing p, RPCO's runtime decreases. Lower values of p lead to more uncorrelated packets, which leads to a blurred graph structure in the sense of more homogeneous edge weights. The Spectral Clustering subroutine tends to return larger clusters on a blurred graph. When not all clusters can be built, RPCO recursively re-clusters them. This is more likely with a blurred graph structure than with a sharp graph structure, i.e., a higher correlation between packets.

Although RPCO has the longest runtime, RPCO can be carried out sufficiently fast so that it can be well applied in practice as configured port clusters may be adapted rather on the time scale of minutes than seconds.

IX. Experimental Performance Evaluation

In this section we perform experiments in a hardware testbed to demonstrate the practical feasibility of the proposed concepts and to validate the theoretical results from Section VIII. First, we explain the concept and the testbed setup. Then, we describe the performed experiments.

A. Concept

Figure 11 illustrates the concept for the hardware testbed.

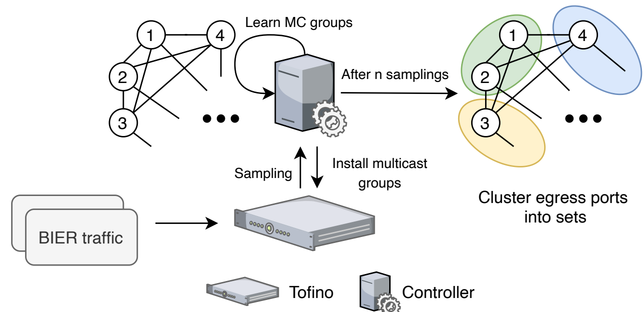


Figure 11: Concept for the hardware evaluation.

The Tofino [49], a P4-programmable switching ASIC, is the core of the hardware testbed. We utilize a Tofino-based Edgecore Wedge 100BF-32X switch [50] with 32 100 Gbit/s ports that runs the adapted BIER implementation as described in Section VI. BIER traffic is sampled at the Tofino with a rate of 0.1%, i.e., every 1000th BIER packet. Sampled packets are sent to the controller and used for the graph embedding as described in Section VII. For 100 Gbit/s incoming multicast traffic, this amounts to 100 Mbit/s which can be efficiently handled by the controller. After 2^{10} samples, the controller applies the optimization heuristic and installs the static multicast groups of the configured port clusters. We measure the average recirculation traffic on the Tofino to assess the effectiveness of the presented optimization heuristics. To that end, packets on the recirculation port are cloned to a separate end host that measures the incoming bandwidth which equals the rate of the recirculation traffic.

B. Traffic Generation

Generating UDP traffic at high rate according to a given distribtion is a difficult task. We leverage Iperf [52] to generate homogeneous UDP traffic on an end host. It is sent to the Tofino which adapts it according to a specified distribution of BIER headers. When the Tofino receives a UDP packet generated by Iperf, it generates a random number between 0 and $2^w - 1$. The generated random number is then used as index to a match-action table that maps the random number to a BIER header (see Figure 12). Then, the header of the UDP packet is substituted by the BIER header indicated in the table. Thereby, a UDP packet stream with any distribution of BIER headers can be generated.

The match-action tables is populated a priori by a controller which has sampled 2^w BIER headers according to the traffic model in Section VIII-A1 for a given set of generating port clusters and a port correlation p. As a result, the Tofino turns homogeneous UDP traffic into BIER traffic whose headers follow a desired distribution.

C. Experiment

We validate our hardware implementation by conducting the same experiments as in Section VIII-B2b. Thus, the traffic

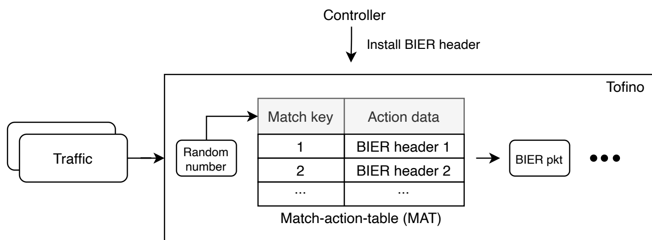


Figure 12: A match-action table is used to turn homogeneous UDP traffic into BIER traffic with headers following a desired distribution.

model consists of six overlapping generating port clusters of size 12, 10, 8, 8, 6, and 4. We choose port correlation $p = 0.9$, and use $w = 14$ to install 2^w sampled BIER headers of that distribution in the match-action table on the Tofino. We generate 5 Gbit/s UDP traffic via Iperf and send it to the Tofino which turns it into BIER traffic with the desired header distribution. We perform 5 runs per experiment and report average values.

The controller samples the BIER traffic and computes optimized port clusters for configuration on the Tofino. Thereby, different port clustering methods and different numbers m_{max} of usable static multicast groups are considered. Figure 13 shows the average recirculation traffic in Gbit/s.

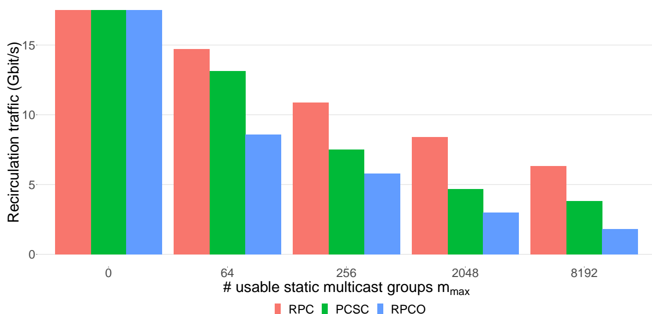


Figure 13: Average recirculation traffic for RPC, PCSC and RPCO and different numbers m_{max} of usable static multicast groups; the traffic model has six overlapping generating port clusters and port correlation $p = 0.9$; the results are to be compared with those in Figure 9(b).

If no static multicast group is available ($m_{max} = 0$), efficient BIER forwarding is essentially disabled, and the observed behaviour is the same as the one for simple BIER forwarding. Therefore, packets with 4.5 ports on average require 3.5 recirculations on average, which results in $3.5 \cdot 5$ Gbit/s $= 17.5$ Gbit/s recirculation traffic. This closely matches the results of Section VIII-B2b. An increasing number m_{max} of usable static multicast groups decreases the average number of recirculations per packet and therefore the recirculation traffic. Again, PCSC and RPCO clearly outperform RPC and RPCO performs better than PCSC (for $m_{max} > 0$). In fact, for $m_{max} = 8192$, RPCO reduces the recirculation traffic by 71% compared to RPC and 52% compared to PCSC. The experimental results in Figure 13 are in line with the simulation results in Figure 9(b) as they show the same proportions.

We performed this experiment with only 5 Gbit/s incoming traffic due to the lack of a fast generator for contant bit rate traffic. However, efficient BIER forwarding runs at line rate at the Tofino[4], i.e., it is capable of handling 32×100 Gbit/s incoming traffic.

X. Conclusion

Bit Index Explicit Replication (BIER) forwards multicast traffic without signalling and states within BIER domains. Thereby, it greatly improves scalability for multicast in core networks. However, a simple implementation of that concept implies iterative packet transmission so that capacity cannot be saved [2] on a single switch. In this paper we presented efficient BIER forwarding with static multicast groups such that a BIER packet can be sent to multiple next-hops in a single pipeline iteration. To that end, we configure port clusters on the switch and install all combinations of ports within each port cluster as static multicast group. Simple match-action operations choose the appropriate port clusters and therein the right static multicast group so that packets are transmitted to multiple next-hops in a single iteration step. As a result, a BIER packet can be processed in high-speed with a single or at most a few iteration steps. We demonstrated by simulation that randomly selected disjoint equal-size configured port clusters can decrease the required recirculations by 90% with only 1024 static multicast groups on a 32 port switch with 32 next-hops (Section VI-C) compared to simple iterative BIER forwarding. Further, we presented port clustering algorithms based on Spectral Clustering which learn the current BIER traffic pattern and compute port clusters for configuration. Recursive Port Clustering with Overlap (RPCO) reduces the required recirculations by up to 96% compared to randomly selected port clusters (Section VIII). We implemented efficient BIER forwarding on the Edgecore Wedge 100BF-32X, a 32 100 Gbit/s port high-performance P4 switch, and validated the simulation results in a hardware testbed.

The work comes with a few byproducts. We developed efficient BIER forwarding for data plane programming with the Tofino ASIC. Other switch architectures will also face the challenge to determine outgoing ports of a BIER packet within short time and can benefit from the presented algorithms. We proposed a traffic model for the outgoing ports of multicast traffic on a switch for evaluation purposes. Future work may validate that traffic model based on measured data. Finally, we developed a simple method for data plane programming to modify traffic such that its headers correspond to a specific distribution. This may also be useful in other experimental work.

References

[1] I. Wijnands *et al.*, *RFC 8279: Multicast Using Bit Index Explicit Replication (BIER)*, https://datatracker.ietf.org/doc/rfc8279/, Nov. 2017.

[4]Every P4 program that compiles for the Tofino runs at line rate.

[2] D. Merling *et al.*, "Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4," *IEEE Access*, vol. 9, 2021.
[3] S. Islam *et al.*, "A Survey on Multicasting in Software-Defined Networking," *IEEE Communications Surveys Tutorials (COMST)*, vol. 20, 2018.
[4] Z. Al-Saeed *et al.*, "Multicasting in Software Defined Networks: A Comprehensive Survey," *Journal of Network and Computer Applications (JNCA)*, vol. 104, 2018.
[5] M. Shahbaz *et al.*, "Elmo: Source Routed Multicast for Public Clouds," in *ACM SIGCOMM*, 2019.
[6] A. Iyer *et al.*, "Avalanche: Data Center Multicast using Software Defined Networking," in *International Conference on Communication Systems and Networks*, 2014.
[7] W. Cui *et al.*, "Scalable and Load-Balanced Data Center Multicast," in *IEEE GLOBECOM*, 2015.
[8] X. Li *et al.*, "Scaling IP Multicast on Datacenter Topologies," in *ACM CoNEXT*, 2013.
[9] X. Zhang *et al.*, "A Centralized Optimization Solution for Application Layer Multicast Tree," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 14, 2017.
[10] K. Mokhtarian *et al.*, "Minimum-delay multicast algorithms for mesh overlays," *IEEE/ACM Transactions on Networking*, vol. 23, 2015.
[11] S.-H. Shen, "Efficient SVC Multicast Streaming for Video Conferencing With SDN Control," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 16, 2019.
[12] M. A. Kaafar *et al.*, "A Locating-First Approach for Scalable Overlay Multicast," in *IEEE INFOCOM*, 2006.
[13] R. Boivie, N. Feldman, and C. Metz, "Small Group Multicast: A New Solution for Multicasting on the Internet," *IEEE Internet Computing*, vol. 4, 2000.
[14] A. Boudani and B. Cousin, "sem: A new small group multicast routing protocol," in *International Conference on Telecommunications (ICT)*.
[15] W. K. Jia *et al.*, "A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 31, 2013.
[16] C. A. S. Oliveira *et al.*, "Steiner Trees and Multicast," *Mathematical Aspects of Network Routing Optimization*, vol. 53, 2011.
[17] L. H. Huang *et al.*, "Scalable and Bandwidth-Efficient Multicast for Software-Defined Networks," in *IEEE GLOBECOM*, 2014.
[18] J.-R. Jiang *et al.*, "Constructing Multiple Steiner Trees for Software-Defined Networking Multicast," in *Conference on Future Internet Technologies*, 2016.
[19] Z. Hu *et al.*, "Multicast Routing with Uncertain Sources in Software-Defined Network," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2016.
[20] S. Zhou *et al.*, "Cost-Efficient and Scalable Multicast Tree in Software Defined Networking," in *Algorithms and Architectures for Parallel Processing*, 2015.
[21] S.-H. Shen *et al.*, "Reliable Multicast Routing for Software-Defined Networks," in *IEEE INFOCOM*, 2015.
[22] B. Ren *et al.*, "The Packing Problem of Uncertain Multicasts," *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.
[23] S. H. Shen, L. H. Huang, D. N. Yang, and W. T. Chen, "Reliable Multicast Routing for Software-Defined Networks," in *IEEE INFOCOM*, 2015.
[24] M. Popovic *et al.*, "Performance Comparison of Node-Redundant Multicast Distribution Trees in SDN Networks," *International Conference on Networked Systems*, 2017.
[25] D. Kotani *et al.*, "A Multicast Tree Management Method Supporting Fast Failure Recovery and Dynamic Group Membership Changes in OpenFlow Networks," *Journal of Information Processing (JIP)*, vol. 24, 2016.
[26] T. Pfeiffenberger *et al.*, "Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow," in *IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2015.
[27] J. Rückert *et al.*, "Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks," *Journal of Network and Systems Management (JNSM)*, vol. 23, 2015.
[28] J. Rueckert *et al.*, "Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, 2016.
[29] T. Humernbrum *et al.*, "Towards Efficient Multicast Communication in Software-Defined Networks," in *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2016.
[30] Y.-D. Lin *et al.*, "Scalable Multicasting with Multiple Shared Trees in Software Defined Networking," *Journal of Network and Computer Applications (JNCA)*, vol. 78, 2017.
[31] M. J. Reed *et al.*, "Stateless Multicast Switching in Software Defined Networks," in *IEEE International Conference on Communications (ICC)*, 2016.
[32] W. Braun *et al.*, "Demo: Scalable and Reliable Software-Defined Multicast with BIER and P4," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2017.
[33] D. Merling *et al.*, "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast," *Journal of Network and Computer Applications (JNCA)*, vol. 169, 2020.
[34] p4lang, *Behavioral-model*, https://github.com/p4lang/behavioral-model, Accessed: 01.28.2021, 2021.
[35] A. Bas, *BMv2 Throughput*, https://github.com/p4lang/behavioral-model/issues/537\#issuecomment-360537441, Jan. 2018.
[36] A. Giorgetti *et al.*, "First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting," in *IEEE European Conference on Networks and Communications (EuCNC)*, 2017.
[37] ——, "Bit Index Explicit Replication (BIER) Multicasting in Transport Networks," in *International Conference on Optical Network Design and Modeling (ONDM)*, 2017.
[38] Y. Desmouceaux *et al.*, "Reliable Multicast with B.I.E.R.," *Journal of Communications and Networks*, vol. 20, 2018.
[39] T. Eckert *et al.*, *Traffic Engineering for Bit Index Explicit Replication BIER-TE*, http://tools.ietf.org/html/draft-eckert-bier-te-arch, Nov. 2017.
[40] T. Eckert and B. Xu, *Carrier Grade Minimalist Multicast (CGM2) using Bit Index Explicit Replication (BIER) with Recursive BitString Structure (RBS) Addresses*, https://datatracker.ietf.org/doc/html/draft-eckert-bier-cgm2-rbs-01, Feb. 2022.
[41] W. Braun *et al.*, "Performance Comparison of Resilience Mechanisms for Stateless Multicast using BIER," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017.
[42] J. B. MacQueen, "Some Methods for Classification and Analysis of MultiVariate Observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, University of California Press, 1967.
[43] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.
[44] U. v. Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing*, vol. 17, 2007.
[45] H. Chen, M. McBride, S. Lindner, M. Menth, A. Wang, G. Mishra, Y. Liu, Y. Fan, L. Liu, and X. Liu, "BIER Fast ReRoute," Internet Engineering Task Force, Internet-Draft

draft-chen-bier-frr-04, Jan. 2022, Work in Progress, 31 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-chen-bier-frr-04.

[46] D. Merling, S. Lindner, and M. Menth, "Comparison of Fast-Reroute Mechanisms for BIER-Based IP Multicast," in *2020 Seventh International Conference on Software Defined Systems (SDS)*, 2020.

[47] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, 2014.

[48] F. Hauser, M. Haeberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," *currently under submission*, 2021.

[49] Intel, *Intel Tofino*, https://www.intel.de/content/www/de/de/products/network-io/programmable-ethernet-switch/tofino-series.html, 2021.

[50] Edge-Core Networks, *Wedge100BF-32X/65X Switch*, https://www.edge-core.com/_upload/images/2021-048-DCS800_Wedge100BF-32X-DS-R08.pdf, 2021.

[51] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[52] iperf2 team, *iperf*, https://iperf.fr.

1.9 A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research

A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research

Frederik Hauser[a], Marco Häberle[a], Daniel Merling[a], Steffen Lindner[a], Vladimir Gurevich[b], Florian Zeiger[c], Reinhard Frank[c], Michael Menth[a]

[a] *University of Tuebingen, Department of Computer Science, Chair of Communication Networks, Tuebingen, Germany*
[b] *Intel, Barefoot Division (BXD), United States of America*
[c] *Siemens AG, Corporate Technology, Munich, Germany*

Abstract

Programmable data planes allow users to define their own data plane algorithms for network devices including appropriate data plane application programming interfaces (APIs) which may be leveraged by user-defined software-defined networking (SDN) control. This offers great flexibility for network customization, be it for specialized, commercial appliances, e.g., in 5G or data center networks, or for rapid prototyping in industrial and academic research. Programming protocol-independent packet processors (P4) has emerged as the currently most widespread abstraction, programming language, and concept for data plane programming. It is developed and standardized by an open community, and it is supported by various software and hardware platforms.

In the first part of this paper we give a tutorial of data plane programming models, the P4 programming language, architectures, compilers, targets, and data plane APIs. We also consider research efforts to advance P4 technology. In the second part, we categorize a large body of literature of P4-based applied research into different research domains, summarize the contributions of these papers, and extract prototypes, target platforms, and source code availability. For each research domain, we analyze how the reviewed works benefit from P4's core features. Finally, we discuss potential next steps based on our findings.

Keywords: P4, SDN, programmable data planes

Email addresses: `frederik.hauser@uni-tuebingen.de` (Frederik Hauser), `marco.haeberle@uni-tuebingen.de` (Marco Häberle), `daniel.merling@uni-tuebingen.de` (Daniel Merling), `steffen.lindner@uni-tuebingen.de` (Steffen Lindner), `vladimir.gurevich@intel.com` (Vladimir Gurevich), `florian.zeiger@siemens.com` (Florian Zeiger), `reinhard.frank@siemens.com` (Reinhard Frank), `menth@uni-tuebingen.de` (Michael Menth)

1. Introduction

Traditional networking devices such as routers and switches process packets using data and control plane algorithms. Users can configure control plane features and protocols, e.g., via CLIs, web interfaces, or management APIs, but the underlying algorithms can be changed only by the vendor. This limitation has been broken up by SDN and even more by data plane programming.

SDN makes network devices programmable by introducing an API that allows users to bypass the built-in control plane algorithms and to replace them with self-defined algorithms. Those algorithms are expressed in software and typically run on an SDN controller with an overall view of the network. Thereby, complex control plane algorithms designed for distributed control can be replaced by simpler algorithms designed for centralized control. This is beneficial for use cases that are demanding with regard to flexibility, efficiency and security, e.g., massive data centers or 5G networks.

Programmable data planes enable users to implement their own data plane algorithms on forwarding devices. Users, e.g., programmers, practitioners, or operators, may define new protocol headers and forwarding behavior, which is without programmable data planes only possible for a vendor. They may also add data plane APIs for SDN control.

Data plane programming changes the power of the users as they can build custom network equipment without any compromise in performance, scalability, speed, or power on appropriate platforms. There are different data plane programming models, each with many implementations and programming languages. Examples are Click [1], VPP [2], NPL [3], and SDNet [4].

Programming protocol-independent packet processors (P4) is currently the most widespread abstraction, programming language, and concept for data plane programming. First published as a research paper in 2014 [5], it is now developed and standardized in the P4 Language Consortium, it is supported by various software- and hardware-based target platforms, and it is widely applied in academia and industry.

In the following, we clarify the contribution of this survey, point out its novelty, explain its organization, and provide a table with acronyms frequently used in this work.

1.1. Contributions

This survey pursues two objectives. First, it provides a comprehensive introduction and overview of P4. Second, it surveys publications describing applied research based on P4 technology. Its main contributions are the following:

- We explain the evolution of data plane programming with P4, relate it to prior developments such as SDN, and compare it to other data plane programming models.
- We give an overview of data plane programming with P4. It comprises the P4 programming language, architectures, compilers, targets, and data

plane APIs. These sections do not only include foundations but also present related work on advancements, extensions, or experiences.

- We summarize research efforts to advance P4 data planes. It comprises optimization of development and deployment, testing and debugging, research on P4 targets, and advances on control plane operation.
- We analyze a large body of literature considering P4-based applied research. We categorize 245 research papers into different application domains, summarize their key contributions, and characterize them with respect to prototypes, target platforms, and source code availability. For each research domain, we analyze how the reviewed works benefit from P4's core features.

We consider publications on P4 that were published until the end of 2020 and selected paper from 2021. Beside journal, conference, and workshop papers, we also include contents from standards, websites, and source code repositories. The paper comprises 519 references out of which 377 are scientific publications: 73 are from 2017 and before, 66 from 2018, 113 from 2019, 116 from 2020, and 9 from 2021.

1.2. Novelty

There are numerous surveys on SDN published in 2014 [6, 7], 2015 [8, 9, 10], and 2016 [11, 12] as well as surveys on OpenFlow (OF) from 2014 [13, 14, 15]. Only one of them [12] mentions P4 in a single sentence. Two surveys of data plane programming from 2015 [10, 9] were published shortly after the release of P4, one conference paper from 2018 [16] and a survey from 2019 [17] present P4 just as one among other data plane programming languages. Likewise, Michel et al. [18] gives an overview of data plane programming in general and P4 is one among other examined abstractions and programming languages. Our survey is dedicated to P4 only. It covers more details of P4 and a many more papers of P4-based applied research which have mostly emerged only within the last two years.

A recent survey focusing on P4 data plane programming has been published in [19]. The authors introduce data plane programming with P4, review 33 research works from four research domains, and discuss research issues. Another recent technical report [20] reviews 150 research papers from seven research domains. While typical research areas of P4 are covered, others (e.g., industrial networking, novel routing and forwarding schemes, and time-sensitive networking) are not part of the literature review. The different aspects of P4, e.g., the programming language, architectures, compilers, targets, data plane APIs, and their advancements are not treated in the paper. In addition, a survey solely focusing on P4 for network security [21] was recently published. Gao et al. introduce the P4 language and review 60 research works in the field of network security applications. They analyze the core idea of the reviewed literature and point out limitations. Finally, a short comparison on P4 targets regarding

throughput, delay, jitter, resource constraints, flexibility and proportion in the research literature is given. In contrast to the mentioned surveys on P4, we cover a greater level of detail of P4 technology and their advancements, and our literature review is more comprehensive.

1.3. Paper Organization

Figure 1 depicts the structure of this paper which is divided into two main parts: an *overview of P4* and a *survey of research publications*.

In the first part, Section 2 gives an introduction to network programmability. We describe the development from traditional networking and SDN to data plane programming and present the two most common data plane programming models. In Section 3, we give a technology-oriented tutorial of P4 based on its latest version $P4_{16}$. We introduce the P4 programming language and describe how user-provided P4 programs are compiled and executed on P4 targets. Section 4 presents the concept of P4 architectures as intermediate layer between the P4 programs and the targets. We introduce the four most common architectures in detail and describe P4 compilers. In Section 5, we categorize and present platforms that execute P4 programs, so-called P4 targets that are based on software, FPGAs, ASICs, or NPUs. Section 6 gives an introduction to data plane APIs. We describe their functions, present a characterization, introduce the four main P4 data plane APIs that serve as interfaces for SDN controllers, and point out controller use case patterns. In Section 7, we summarize research efforts that aim to improve P4 data plane programming.

The second part of the paper surveys P4-based applied research in communication networks. In Section 8, we classify core features of P4 that make it attractive for the implementation of data plane algorithms. We use these properties in later sections to effectively reason about P4's value for the implementation of various prototypes. We present an overview of the research domains and compile statistics about the included publications. The superordinate research domains are monitoring (Section 9), traffic management and congestion control (Section 10), routing and forwarding (Section 11), advanced networking (Section 12), network security (Section 13), and miscellaneous (Section 14) to cover additional, different topics. Each category includes a table to give a quick overview of the analyzed papers with regard to prototype implementations, target platforms, and source code availability. At the end of each section, we analyze how the reviewed works benefit from P4's core features.

In Section 15 we discuss insights from this survey and give an outlook on potential next steps. Section 16 concludes this work.

1.4. List of Acronyms

The following acronyms are used in this paper.

ACL access control list

ALU arithmetic logic unit

API application programming interface

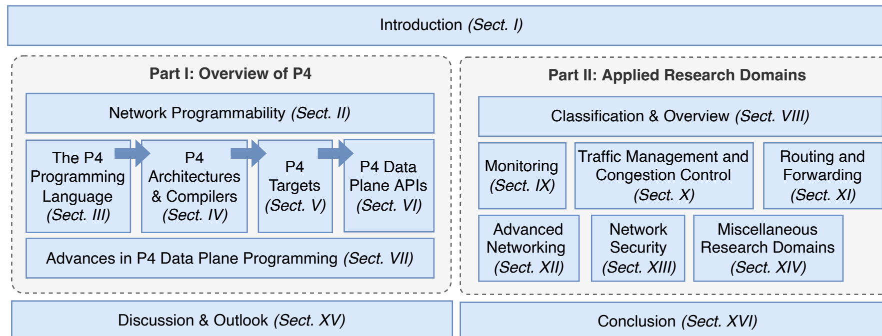


Figure 1: The paper is organized in two parts: Part I gives an overview on P4; Part II reviews P4-based applied research in communication networks.

AQM	active queue management
ASIC	application-specific integrated circuit
AWW	adjusting advertised windows
bmv2	Behavioral Model version 2
BGP	Border Gateway Protocol
BPF	Berkeley Packet Filter
CLI	command line interface
DAG	directed acyclic graph
DDoS	distributed denial of service
DPI	deep packet inspection
DPDK	Data Plane Development Kit
DSL	domain-specific language
eBPF	Extended Berkeley Packet Filter
ECN	Explicit Congestion Notification
FPGA	field programmable gate array
FSM	finite state machine
GTP	GPRS tunneling protocol
HDL	hardware description language
HLIR	high-level intermediate representation
IDE	integrated development environment
IDL	Intent Definition Language
IDS	intrusion detection system
INT	in-band network telemetry

LDWG	Language Design Working Group
LPM	longest prefix matching
LUT	look up table
MAT	match-action-table
ML	machine learning
NDN	named data networking
NF	network function
NFP	network flow processing
NFV	network function virtualization
NIC	network interface card
NPU	network processing unit
ODM	original design manufacturer
ODP	Open Data Plane
OEM	original equipment manufacturer
OF	OpenFlow
ONF	Open Networking Foundation
OVS	Open vSwitch
PISA	Protocol Independent Switching Architecture
PSA	Portable Switch Architecture
REG	register
RPC	remote procedure call
RTL	register-transfer level
SDK	software development kit
SDN	software-defined networking
SF	service function
SFC	service function chain
SRAM	static random-access memory
TCAM	ternary content-addressable memory
TSN	Time-Sensitive Networking
TNA	Tofino Native Architecture
uBPF	user-space BPF
VM	virtual machine
VNF	virtual network function
VPP	Vector Packet Processors
WG	working group
XDP	eXpress Data Path

2. Network Programmability

In this section, we first define the notion of network programmability and related terms. Then, we discuss control plane programmability and data plane programming, elaborate on data plane programming models, and point out the benefits of data plane programming.

2.1. Definition of Terms

We define *programmability* as the ability of the software or the hardware to execute an externally defined processing algorithm. This ability separates programmable entities from *flexible* (or *configurable*) ones; the latter only allow changing different parameters of the internally defined algorithm which stays the same.

Thus, the term *network programmability* means the ability to define the processing algorithm executed in a network and specifically in individual processing nodes, such as switches, routers, load balancers, etc. It is usually assumed that no special processing happens in the links connecting network nodes. If necessary, such processing can be described as if it takes place on the nodes that are the endpoints of the links or by adding a "bump-in-the-wire" node with one input and one output.

Traditionally, the algorithms, executed by telecommunication devices, are split into three distinct classes: the data plane, the control plane, and the management plane. Out of these three classes, the management plane algorithms have the smallest effect on both the overall packet processing and network behavior. Moreover, they have been programmable for decades, e.g., SNMPv1 was standardized in 1988 and created even earlier than that. Therefore, management plane algorithms will not be further discussed in this section.

True network programmability implies the ability to specify and change both the control plane and data plane algorithms. In practice this means the ability of network operators (users) to define both data and control plane algorithms on their own, without the need to involve the original designers of the network equipment. For the network equipment vendors (who typically design their own control plane anyway), network programmability mostly means the ability to define data plane algorithms without the need to involve the original designers of the chosen packet processing application-specific integrated circuit (ASIC).

Network programmability is a powerful concept that allows both the network equipment vendors and the users to build networks ideally suited to their needs. In addition, they can do it much faster and often cheaper than ever before and without compromising the performance or quality of the equipment.

For a variety of technical reasons, different layers became programmable at different point in time. While the management plane became programmable in the 1980s, control plane programmability was not achieved until late 2000s to early 2010s and a programmable switching ASICs did not appear till the end of 2015.

Thus, despite the focus on data plane programmability, we will start by discussing control plane programmability and its most well-known embodiment,

called software-defined networking (SDN). This discussion will also better prepare us to understand the significance of data plane programmability.

2.2. Control Plane Programmability and SDN

Traditional networking devices such as routers or switches have complex data and control plane algorithms. They are built into them and generally cannot be replaced by the users. Thus, the functionality of a device is defined by its vendor who is the only one who can change it. In industry parlance, vendors are often called original equipment manufacturers (OEMs).

Software-defined networking (SDN) was historically the first attempt to make the devices, and *specifically their control plane*, programmable. On selected systems, device manufacturers allowed users to bypass built-in control plane algorithms so that the users can introduce their own. These algorithms could then directly supply the necessary forwarding information to the data plane which was still non-replaceable and remained under the control of the device vendor or their chosen silicon provider.

For a variety of technical reasons, it was decided to provide an APIs that could be called remotely and that is how SDN was born. Figure 2 depicts SDN in comparison to traditional networking. Not only the control plane became programmable, but it also became possible to implement network-wide control plane algorithms in a centralized controller. In several important use cases, such as tightly controlled, massive data centers, these centralized, network-wide algorithms proved to be a lot simpler and more efficient, than the traditional algorithms (e.g. Border Gateway Protocol (BGP)) designed for decentralized control of many autonomous networks.

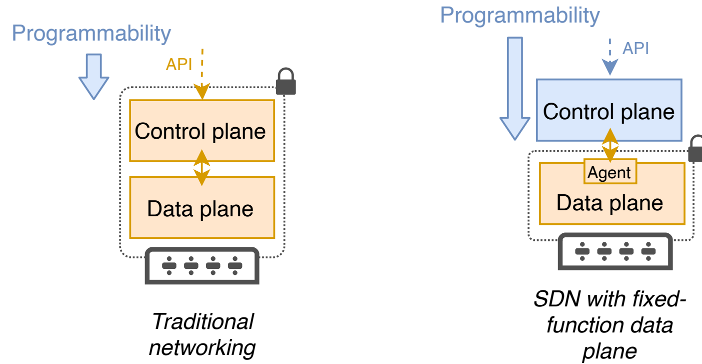
The effort to standardize this approach resulted in the development of OpenFlow (OF) [22]. The hope was that once OF standardized the messaging API to control the data plane functionality, SDN applications will be able to leverage the functions offered by this API to implement network control. There is a huge body of literature giving an overview of OF [13, 14, 15] and SDN [6, 7, 8, 9, 11, 10, 12].

However, it soon became apparent that OF assumed a specific data plane functionality which was not formally specified. Moreover, the specific data plane, that served as the basis for OF, could not be changed. It executed the sole, although relatively flexible, algorithm defined by the OF specifications.

In part, it was this realization that led to the development of modern data plane programming that we discuss in the following section.

2.3. Data Plane Programming

As mentioned above, data plane programmability means that the data plane with its algorithms can be defined by the users, be they network operators or equipment designers working with a packet processing ASIC. In fact, data plane programmability existed during most of the networking industry history because data plane algorithms were typically executed on general-purpose CPUs. It is



(a) With traditional networking, programmability is limited to configuration of functionality via an API.

(b) SDN with fixed-function data planes allows full programmability of the control plane.

Figure 2: Distinction between traditional networking and SDN with fixed-function data planes.

only with the advent of high-speed links, exceeding the CPU processing capabilities, and the subsequent introduction of packet processing (switching) ASICs that data plane programmability (or lack thereof) became an issue.

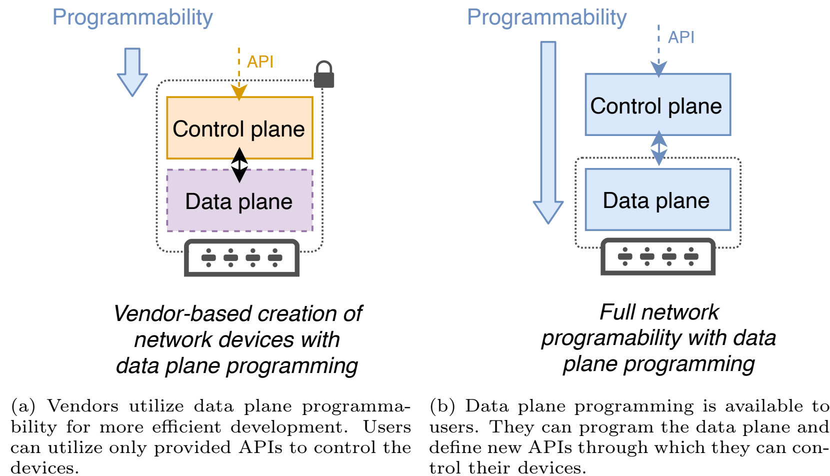
The data plane algorithms are responsible for processing all the packets that pass through a telecommunication system. Thus, they ultimately define the functionality, performance, and the scalability of such systems. Any attempt to implement data plane functionality in the control plane typically leads to significant performance degradation. When data plane programming is provided to users, it qualitatively changes their power. They can build custom network equipment without any compromise in performance, scalability, speed, or energy consumption.

For custom networks, new control planes and SDN applications can be designed and for them users can design data plane algorithms that fit them ideally. Data plane programming does not necessarily imply any provision of APIs for users nor does it require support for outside control planes as in OF. Device vendors might still decide to develop a proprietary control plane and use data plane programming only for their own benefit without necessarily making their systems more open (although many do open their systems now). Figure 3 visualizes both options.

Four surveys from [10, 9, 16, 17] give an overview on data plane programming, but do not set a particular focus to P4.

2.4. Data Plane Programming Models

Data plane algorithms can and often are expressed using standard programming languages. However, they do not map very well onto specialized hardware such as high-speed ASICs. Therefore, several data plane models have been proposed as abstractions of the hardware. Data plane programming languages are tailored to those data plane models and provide ways to express algorithms



(a) Vendors utilize data plane programmability for more efficient development. Users can utilize only provided APIs to control the devices.

(b) Data plane programming is available to users. They can program the data plane and define new APIs through which they can control their devices.

Figure 3: Different usages of data plane programmability.

for them in an abstract way. The resulting code is then compiled for execution on a specific packet processing node supporting the respective data plane programming model.

Data flow graph abstractions and the Protocol Independent Switching Architecture (PISA) are examples for data plane models. We give an overview of the first and elaborate in-depths on the second as PISA is the data plane programming model for P4.

2.4.1. Data Flow Graph Abstractions

In these data plane programming models, packet processing is described by a directed graph. The nodes of the graph represent simple, reusable primitives that can be applied to packets, e.g., packet header modifications. The directed edges of the graph represent packet traversals where traversal decisions are performed in nodes on a per-packet basis. Figure 4 shows an exemplary graph for IPv4 and IPv6 packet forwarding.

Examples for programming languages that implement this data plane programming model are Click [1], Vector Packet Processors (VPP) [2], and BESS [23].

2.4.2. Protocol-Independent Switching Architecture (PISA)

Figure 5 depicts the PISA. It is based on the concept of a programmable match-action pipeline that well matches modern switching hardware. It is a generalization of reconfigurable match-action tables (RMTs) [24] and disaggregated reconfigurable match-action tables (dRMTs) [25].

PISA consists of a programmable parser, a programmable deparser, and a programmable match-action pipeline in between consisting of multiple stages.

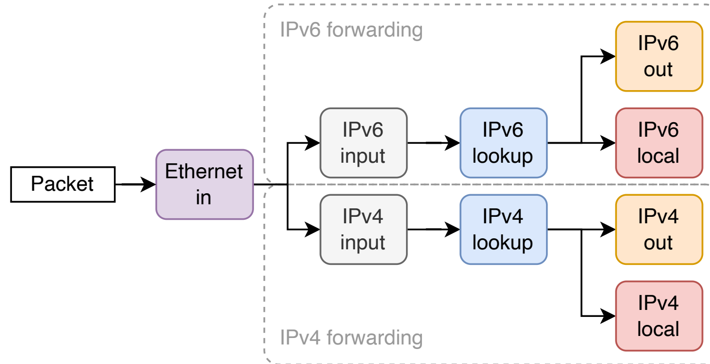


Figure 4: Example graph showing how data flow graph abstractions are applied to implement IPv4 and IPv6 forwarding.

- The *programmable parser* allows programmers to declare arbitrary headers together with a finite state machine that defines the order of the headers within packets. It converts the serialized packet headers into a well-structured form.
- The *programmable match-action pipeline* consists of multiple match-action units. Each unit includes one or more match-action-tables (MATs) to match packets and perform match-specific actions with supplied action data. The bulk of a packet processing algorithm is defined in the form of such MATs. Each MAT includes matching logic coupled with the memory (static random-access memory (SRAM) or ternary content-addressable memory (TCAM)) to store lookup keys and the corresponding action data. The action logic, e.g., arithmetic operations or header modifications, is implemented by arithmetic logic units (ALUs). Additional action logic can be implemented using stateful objects, e.g., counters, meters, or registers, that are stored in the SRAM. A control plane manages the matching logic by writing entries in the MATs to influence the runtime behavior.
- In the *programmable deparser*, programmers declare how packets are serialized.

A packet, processed by a PISA pipeline, consists of packet payload and packet metadata. PISA only processes packet metadata that travels from the parser all the way to the deparser but not the packet payload that travels separately.

Packet metadata can be divided into packet headers, user-defined and intrinsic metadata.

- *Packet headers* is metadata that corresponds to the network protocol headers. They are usually extracted in the parser, emitted in the deparser or both.

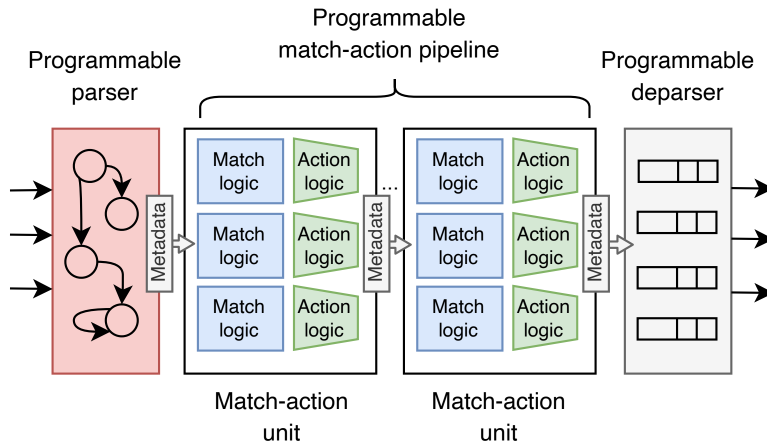


Figure 5: The Protocol-Independent Switch Architecture (PISA) contains a programmable parser, a programmable match-action pipeline, and a programmable deparser.

- *Intrinsic metadata* is metadata that relates to the fixed-function components. P4-programmable components may receive information from the fixed-function components by reading the intrinsic metadata they produce or control their behavior by setting the intrinsic metadata they consume.
- *User-defined metadata* (often referred as simply *metadata*) is a temporary storage, similar to local variables in other programming languages. It allows the developers to add information to packets that can be used throughout the processing pipeline.

All metadata, be it packet headers, user-defined or intrinsic metadata is *transient*, meaning that it is discarded when the corresponding packet leaves the processing pipeline (e.g., is sent out of an egress port or dropped).

PISA provides an abstract model that is applied in various ways to create concrete architectures. For example, it allows specifying pipelines containing different combinations of programmable components, e.g., a pipeline with no parser or deparser, a pipeline with two parsers and deparsers, and additional match-action pipelines between them. PISA also allows for specialized components that are required for advanced processing, e.g., hash/checksum calculations. Besides the programmable components of PISA, switch architectures typically also include configurable fixed-function components. Examples are ingress/egress port blocks that receive or send packets, packet replication engines that implements multicasting or cloning/mirroring of packets, and traffic managers, responsible for packet buffering, queuing, and scheduling.

The fixed-function components communicate with the programmable ones by generating and/or consuming intrinsic metadata. For example, the ingress port block generates ingress metadata that represents the ingress port number that might be used within the match-action units. To output a packet, the

match-action units generates intrinsic metadata that represents an egress port number; this intrinsic metadata is then consumed by the traffic manager and/or egress port block.

Figure 6 depicts a typical switch architecture based on PISA. It comprises a programmable ingress and egress pipeline and three fixed-function components: an ingress block, an egress block, and a packet replication engine together with a traffic manager between ingress and egress pipeline.

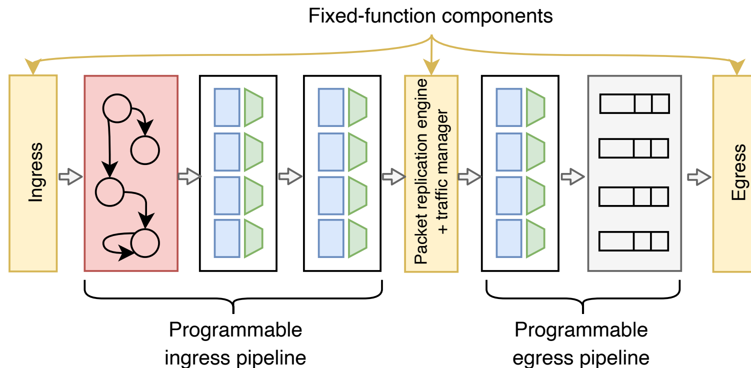


Figure 6: Exemplary switch architecture based on PISA with the ingress and egress pipeline as programmable parts. The ingress, the egress, the packet replication engine, and the traffic manager are fixed-function components.

P4 (Programming Protocol-Independent Packet Processors) [5] is the most widely used domain-specific programming language for describing data plane algorithms for PISA. Its initial idea and name were introduced in 2013 [26] and it was published as a research paper in 2014 [5]. Since then, P4 has been further developed and standardized by the P4 Language Consortium [27] that is part of the Open Networking Foundation (ONF) since 2019. The P4 Language Consortium is managed by a technical steering committee and hosts five working groups (WGs). $P4_{14}$ [28] was the first standardized version of the language. The current specification is $P4_{16}$ [29] which was first introduced in 2016.

Other data plane programming languages for PISA are FAST [30], OpenState [31], Domino [32], FlowBlaze [33], Protocol-Oblivious Forwarding [34], and NetKAT [35]. In addition, Broadcom [3] and Xilinx [4] offer vendor-specific programmable data planes based on match-action tables.

2.5. Benefits

Data plane programmability entails multiple benefits. In the following, we summarize key benefits.

Data plane programming introduces full flexibility to network packet processing, i.e., algorithms, protocols, features can be added, modified, or removed by the user. In addition, programmable data planes can be equipped with a user-defined API for control plane programmability and SDN. To keep complexity low, only components needed for a particular use case might be included

in the code. This improves security and efficiency compared to multi-purpose appliances.

In conjunction with suitable hardware platforms, data plane programming allows network equipment designers and even users to experiment with new protocols and design unique applications; both do no longer depend on vendors of specialized packet-processing ASICs to implement custom algorithms. Compared to long development circles of new silicon-based solutions, new algorithms can be programmed and deployed in a matter of days.

Data plane programming is also beneficial for network equipment developers that can easily create differentiated products despite using the same packet processing ASIC. In addition, they can keep their know-how to themselves without the need to share the details with the ASIC vendor and potentially disclose it to their competitors that will use the same ASIC.

So far, modern data plane programs and programming languages have not yet achieved the degree of portability attained by the general-purpose programming languages. However, expressing data plane algorithms in a high-level language has the potential to make telecommunication systems significantly more target-independent. Also, data plane programming does not require but encourages full transparency. If the source code is shared, all definitions for protocols and behaviors can be viewed, analyzed, and reasoned about, so that data plane programs benefit from community development and review. As a result, users could choose cost-efficient hardware that is well suited for their purposes and run their algorithms on top of it. This trend has been fueled by SDN and is commonly known as network disaggregation.

2.6. Differences Between SDN and P4

SDN introduces *programmability on the control plane*. SDN-capable network devices such as switches include an API allowing that the device-local control plane can be substituted by an external, software-based control plane. This control plane comprises control plane algorithms managing the data plane. The centralized view of an external controller facilitates the implementation of simpler algorithms that may replace complex distributed protocols from legacy network devices. The control plane leverages an API offered by the data plane devices for control. The data plane however merely features fixed functions that can be used and configured by the control plane.

In contrast, P4 is a domain-specific language for data plane programming, i.e., *programmability is extended to the data plane*. Instead of supporting fixed functions only, the functionality of the data plane devices is described by a P4 program that is compiled into target-specific code that can be executed by the programmable network hardware. While the P4 language itself focuses on data plane programmability, P4 targets typically offer APIs so that software-based SDN control planes can manage the runtime behavior of those data plane devices.

3. The P4 Programming Language

We give an overview of the P4 programming language. We briefly recap its specification history and describe how P4 programs are deployed. We introduce the P4 processing pipeline and data types. We discuss parsers, match-action controls, and deparsers. Finally, we give an overview of tutorials and guides to P4.

3.1. Specification History

The P4 Language Design Working Group (LDWG) of the P4 Language Consortium has standardized so far two distinct standards of P4: $P4_{14}$ and $P4_{16}$. Table 1 depicts their specification history.

Table 1: Specification history of $P4_{14}$ and $P4_{16}$.

$P4_{14}$	
Version 1.0.2	03/2015
Version 1.1.0	01/2016
Version 1.0.3	11/2016
Version 1.0.4	05/2017
Version 1.0.5	11/2018

$P4_{16}$	
Version 1.0.0	05/2017
Version 1.1.0	11/2018
Version 1.2.0	11/2018
Version 1.2.1	06/2020

The $P4_{14}$ programming language dialect allows the programmers to describe data plane algorithms using a combination of familiar, general-purpose imperative constructs and more specialized declarative ones that provide support for the typical data-plane-specific functionality, e.g., counters, meters, checksum calculations, etc. As a result, the $P4_{14}$ language core includes more than 70 keywords. It further assumed a specific pipeline architecture based on PISA.

Table 2: Core differences between $P4_{14}$ and $P4_{16}$.

	$P4_{14}$	**$P4_{16}$**
Modularity	-	✓
Pipeline architectures	single	multiple
Target-specific functions	-	✓
# of language keywords	>70	<40
Strict typing	-	✓
Nested data structures	-	✓
Declarative constructs	✓	-

$P4_{16}$ has been introduced to address several $P4_{14}$ limitations that became apparent in the course of its use. Those include the lack of means to describe various targets and architectures, weak typing and generally loose semantics (caused, in part, by the above-mentioned mix of imperative and declarative programming constructs), relatively low-level constructs, and weak support for program modularity. The core differences between $P4_{14}$ and $P4_{16}$ are summarized in Table 2.

Support for multiple different targets and pipeline architecture is the major contribution of the $P4_{16}$ standard and is achieved by separating the core language from the specifics of a given architecture, thus making it architecture-agnostic. The structure, capabilities and interfaces of a specific pipeline are now encapsulated into an architecture description, while the architecture- or target-specific functions are accessible through an architecture library, typically provided by the target vendor. The core components are further structured into a small set of language constructs and a core library that is useful for most P4 programs. Compared to $P4_{14}$, $P4_{16}$ introduced strict typing, expressions, nested data structures, several modularity mechanisms, and also removed declarative constructs, making it possible to better reason about the programs, written in the language. Figure 7 illustrates the concept which is subdivided into core components and architecture components.

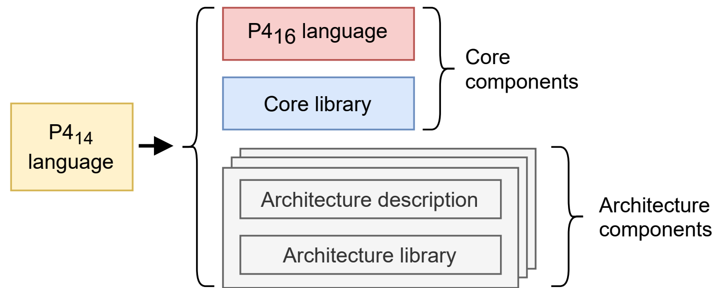


Figure 7: Evolvement from the $P4_{14}$ programming language to the $P4_{16}$ language (similar to [29]). $P4_{14}$ comprised all components as part of the programming language. In $P4_{16}$, the different parts of the programming language are split into core components and architecture components.

Due to the obvious advantages of $P4_{16}$, $P4_{14}$ development has been discontinued, although it is still supported on a number of targets. Therefore, we focus on $P4_{16}$ in the remainder of this paper where P4 implicitly stands for $P4_{16}$.

3.2. Development and Deployment Process

Figure 8 illustrates the development and deployment process of P4 programs.

P4-programmable nodes, so-called P4 targets, are available as software or specialized hardware (see Section 5). They feature packet processing pipelines consisting of both P4-programmable and fixed-function components. The exact structure of these pipelines is target-specific and is described by a corresponding P4 architecture model (see Section 4) which is provided by the manufacturer of the target.

P4 programs are supplied by the user and are implemented for a particular P4 architecture model. They define algorithms that will be executed by the P4-programmable components and their interaction with the ones implemented in the fixed-function logic. The composition of the P4 programs and the fixed-function logic constitutes the full data plane algorithm.

P4 compilers (see Section 4) are also provided by the manufacturers. They translate P4 programs into target-specific code which is loaded and executed by the P4 target.

The P4 compiler also generates a data plane API that can be used by a user-supplied control plane (see Section 6) to manage the runtime behavior of the P4 target.

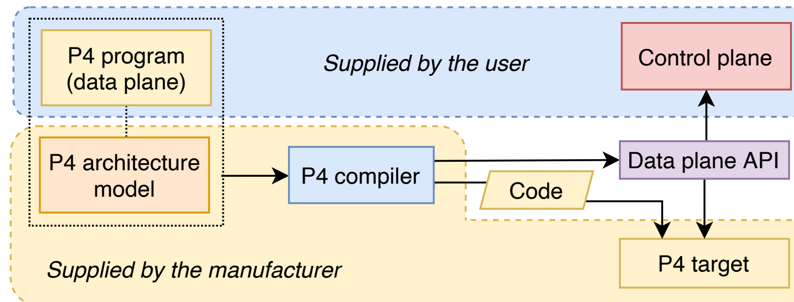


Figure 8: P4 deployment process (similar to [29]): A P4 compiler transforms a P4 program formulated for a particular P4 architecture model into code which is executed by a P4 target. The code provides a data plane API which can be leveraged by a user-supplied control plane.

3.3. Information Flow

$P4_{16}$ adopts PISA's concept of packet metadata. Figure 9 illustrates the information flow in the P4 processing pipeline. It comprises different blocks, where packet metadata (be it headers, user-defined or intrinsic metadata) is used to pass the information between them, therefore representing a uniform interface.

The parser splits up the received packet into individual headers and the remaining payload. Intrinsic metadata from the ingress block, e.g., the ingress port number or the ingress timestamp, is often provided by the hardware and can be made available for further processing. Many targets allow the user metadata to be initialized in the parser as well. Then, the headers and metadata are passed to the match-action pipeline that consists of one or more match-action units. The remaining payload travels separately and cannot be directly affected by the match-action pipeline processing.

While traversing the individual match-action pipeline units, the headers can be added, modified, or removed and additional metadata can be generated.

The deparser assembles the packet back by emitting the specified headers followed by the original packet payload. Packet output is configured with intrinsic metadata that includes information such as a drop flag, desired egress port, queue number, etc.

3.4. Data Types

$P4_{16}$ is a statically typed language that supports a rich set of data types for data plane programming.

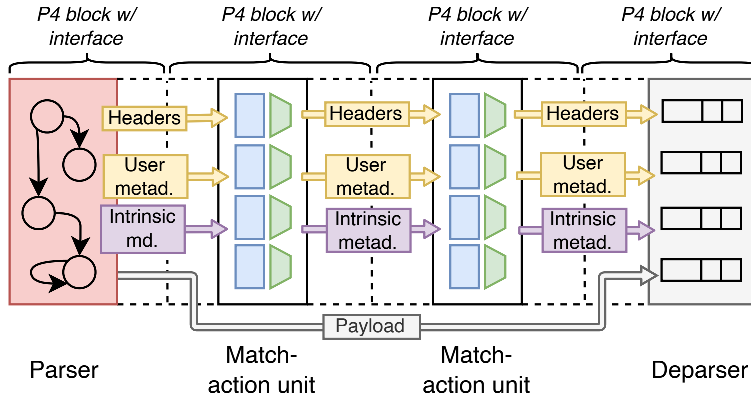


Figure 9: Information flow in the P4 processing pipeline. Metadata (headers, user metadata, intrinsic metadata) transport information between the different P4 blocks of the processing pipeline.

3.4.1. Basic Data Types

$P4_{16}$ includes common basic types such as Boolean (`bool`), signed (`int`), and unsigned (`bit`) integers which are also known as bit strings. Unlike many common programming languages, the size of these integers is specified at *bit* granularity, with a wide range of supported widths. For example, types such as `bit<1>`, `int<3>`, `bit<128>` and wider are allowed.

In addition, P4 supports bit strings of variable width, represented by a special varbit type. For example, IPv4 options can be represented as `varbit<320>` since the size of IPv4 options ranges from zero to 10 32-bit words.

$P4_{16}$ also supports enumeration types that can be serializable (with the actual representation specified as `bit<N>` or `int<N>` during the type definition) or non-serializable, where the type representation is chosen by the compiler and hidden from the user.

3.4.2. Derived Data Types

Basic data types can be composed to construct derived data types. The most common derived data types are `header`, `header stack`, and `struct`.

The `header` data type facilitates the definition of packet protocol headers, e.g., IPv4 or TCP. A header consists of one more fields of the serializable types described above, typically `bit<N>`, serializable `enum`, or `varbit`. A header also has an implicit validity field indicating whether the header is part of a packet. The field is accessible through standard methods such as *setvalid()*, *setInvalid()*, and *isValid()*. Packet parsing starts with all headers being invalid. If the parser determines that a header is present in the packet, the header fields are extracted and the header's validity field is set valid. The standard packet *emit()* method used by a deparser equips packets only with valid headers. Thus, P4 programs can easily add and remove headers by manipulating their validity bits. A sample header declaration is shown in Figure 10.

A `header stack` is used to define repeating headers, e.g., VLAN tags or MPLS labels. It supports special operations allowing headers to be "pushed" onto the stack or "popped" from it.

`Struct` in P4 is a composed data type similar to structs in programming languages like C. Unlike the `header` data type, they can contain fields of any type including other structs, headers, and others.

```
typedef bit<48> macAddr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}
```

Figure 10: Sample declaration of the Ethernet header with the help of a type definition for the MAC addresses used in the header.

3.5. Parsers

Parsers extract header fields from ingress packets into header data and metadata. P4 does not include predefined packet formats, i.e., all required header formats including parsing mechanisms need to be part of the P4 program. Parsers are defined as finite state machine (FSM) with an explicit *Start* state, two ending states (*Accept* and *Reject*), and custom states in between.

Figure 11 depicts the structure of a typical P4 parser for Ethernet, MPLS, IPv4, TCP, and UDP headers. Figure 12 shows the source code fragment of the example parser in a $P4_{16}$ program. The process starts in the *Start* state and switches to the *Ethernet* state. In this state and the following states, information from the packet headers is extracted according to the defined header structure.

State transitions may be either conditional or unconditional. In the given example, the transition from the *Start* state to the *Ethernet* state is unconditional while in the *Ethernet* state the transition to the *MPLS*, *IPv4*, or *Reject* state depends on the value of the *EtherType* field of the extracted Ethernet header. Based on previously parsed header information, any number of further headers can be extracted from the packet. If the header order does not comply with the expected order, a packet can be discarded by switching to the *Reject* state. The parser can also implicitly transition into the *Reject* state in case of a parser exception, e.g., if a packet is too short.

3.6. Match-Action Controls

Match-action controls express the bulk of the packet processing algorithm and resemble traditional imperative programs. They are executed after successful parsing of a packet. In some architectures they are also called match-action pipeline units. In the following, we give an overview of control blocks, actions, and match-action tables.

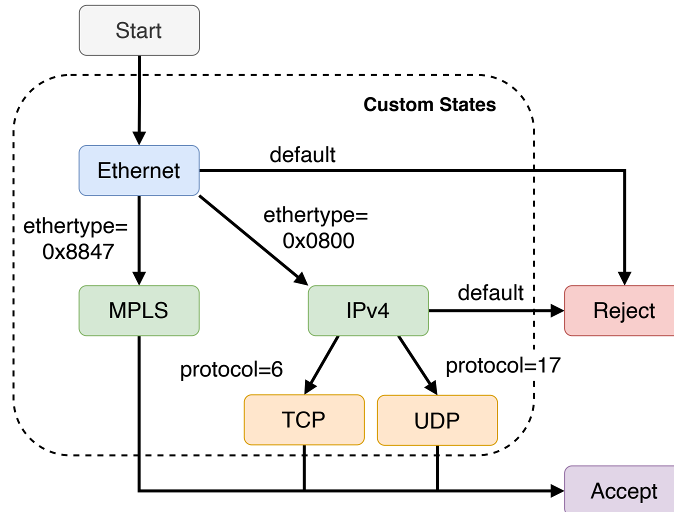


Figure 11: Example for the FSM of a P4 parser that parses packets with Ethernet, MPLS, IPv4, TCP, and UDP headers.

3.6.1. Control Blocks

Control blocks, or just `controls`, are similar to functions in general-purpose languages. They are called by an *apply()* method. They have parameters and can call also other control blocks. The body of a control block contains the definition of resources, such as tables, actions, and externs that will be used for processing. Furthermore, a single *apply()* method is defined that expresses the processing algorithm.

P4 offers statements to express the program flow within a control block. Unlike common programming languages, P4 does not provide any statements that would allow the programmer to create loops. This ensures that all the algorithms that can be coded in P4 can be expressed as directed acyclic graphs (DAGs) and thus are guaranteed to complete within a predictable time interval. Specific control statements include:

- a block statement `{}` that expresses sequential execution of instructions.
- an `if()` statement that expresses an execution predicated on a Boolean condition
- a `switch()` statement that expresses a choice from multiple alternatives
- an `exit()` statement that ends the control flow within a control block and passes the control to the end of the top-level control

Transformations are performed by several constructs, such as

- An assignment statement which evaluates the expression on its right-hand-side and assigns the result to a header or a metadata fields

```
parser SampleParser(packet_in p, out headers h) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        p.extract(h.ethernet);
        transition select(h.ethernet.etherType) {
             0x8847: parse_mpls;
             0x0800: parse_ipv4;
            default: reject;
        };
    }

    state parse_ipv4 {
        p.extract(h.ipv4);
        transition select(h.ipv4.protocol) {
                  6: parse_tcp;
                 17: parse_udp;
            default: accept;
        }
    }

    state parse_udp {
        p.extract(h.udp);
        transition accept;
    }
    /* Other states follow */
}
```

Figure 12: Sample parser implementation of the FSM in Figure 11.

- A match-action operation on a table expressed as the table's `apply()` method
- An invocation of an action or a function that encapsulate a sequence of statements
- An invocation of an extern method that represents special, target- and architecture-specific processing, often involving additional state, preserved between packets

A sample implementation of basic L2 forwarding is provided in Figure 13.

3.6.2. Actions

Actions are code fragments that can read and write packet headers and metadata. They work similarly to functions in other programming languages but have no return value. Actions are typically invoked from MATs. They can receive parameters that are supplied by the control plane as action data in MAT entries.

```
control SampleControl(inout headers h, inout standard_metadata_t
    standard_metadata) {

    action l2_forward(egressSpec_t port) {
        standard_metadata.egress_spec = port;
    }

    table l2 {
        key = {
            h.ethernet.dstAddr: exact;
        }
        actions = {
            l2_forward; drop;
        }
        size = 1024;
        default_action = drop();
    }

    apply {
        if (h.ethernet.isValid()) {
            l2.apply();
        }
    }
}
```

Figure 13: Sample control block implementing basic L2 forwarding.

As in most general-purpose programming languages, the operations are written using expressions and the results are then assigned to the desired header or metadata fields. The operations available in P4 expressions include standard arithmetic and logical operations as well as more specialized ones such as bit slicing (`field[high:low]`), bit concatenation (`field1 ++ field2`), and saturated arithmetic (`|+|` and `|-|`).

Actions can also invoke methods of other objects, such as headers and architecture-specific externs, e.g., counters and meters. Other actions can also be called, similar to nested function calls in traditional programming languages.

Action code is executed sequentially, although many hardware targets support parallel execution. In this case, the compiler can optimize the action code for parallel execution as long as its effects are the same as in case of the sequential execution.

3.6.3. Match-Action Tables (MATs)

MATs are defined within control blocks and invoke actions depending on header and metadata fields of a packet. The structure of a MAT is declared in the P4 program and its table entries are populated by the control plane at runtime. A packet is processed by selecting a matching table entry and invoking the corresponding action with appropriate parameters.

The declaration of a MAT includes the match key, a list of possible actions, and additional attributes.

The match key consists of one or more header or metadata fields (variables), each with the assigned *match type*. The P4 core library defines three standard match types: exact, ternary, and longest prefix matching (LPM). P4 architectures may define additional match types, e.g., the *v1model* P4 architecture extends the set of standard match types with the range and selector match.

The list of possible actions includes the names of all actions that can be executed by the table. These actions can have additional, directional parameters which are provided as action data in table entries.

Additional attributes may include the size of the MAT, e.g., the maximum number of entries that can be stored in a table, a default action for a miss, or static table entries.

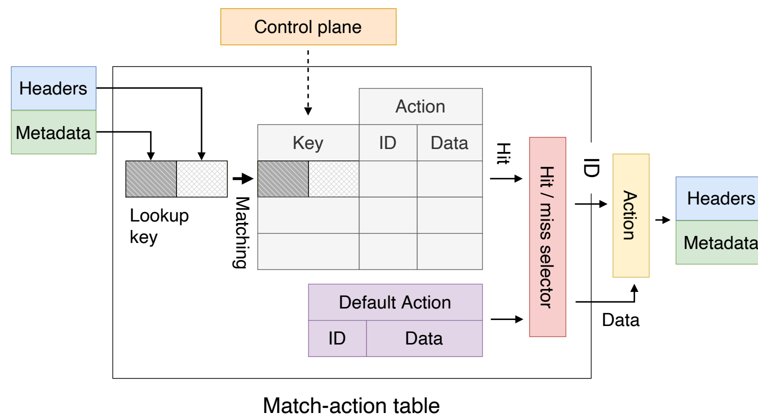


Figure 14: Structure of MATs in P4. Lookup keys are constructed based on packet metadata and used for row matching in the MAT. In case of a hit, the defined action is applied with the specified action data. In case of a miss, the default action is applied.

Figure 14 illustrates the principle of MAT operation. The MAT contains entries with values for match keys, the ID of the corresponding action to be invoked, and action data that serve as parameters for action invocation. For each packet, a lookup key is constructed from the set of header and metadata fields specified in the table definition. It is matched against all entries of the MAT using the rules associated with the individual field's match type. When the first match in the table is found, the corresponding action is called and the action data are passed to the action as directionless parameters. If no match is found in the table, a default action is applied.

As a special case, tables without a specified key always invoke the default action.

3.7. Deparser

The deparser is also defined as a control block. When packet processing by match-action control blocks is finished, the deparser serializes the packet. It reassembles the packet header and payload back into a byte stream so that

the packet can be sent out via an egress port or stored in a buffer. Only valid headers are emitted, i.e., added to the packet. Thus, match-action control blocks can easily add and remove headers by manipulating their validity. Figure 15 provides a sample implementation.

```
control SampleDeparser(packet_out p, in headers h) {
    apply {
        p.emit(h.ethernet);
        p.emit(h.mpls);
        p.emit(h.ipv4);
        /* Normally, a packet can contain either
         * a TCP or a UDP header (or none at all),
         * but should never contain both
         */
        p.emit(h.tcp);
        p.emit(h.udp);
    }
}
```

Figure 15: Sample deparser implementation.

3.8. P4 Tutorials

The P4 Language Consortium provides a GitHub repository with simple programming exercises and a development VM containing all required software [36]. A guide on GitHub lists useful information for P4 newcomers, e.g. demo programs, information about other GitHub repositories, and an overview of P4 [37]. The Networked Systems Group at ETH Zürich provides resources for people who want to learn programming in P4, including lecture slides, references to useful documentation, examples and exercises [38].

4. P4 Architectures & Compilers

We present $P4_{16}$ architectures and introduce P4 compilers.

4.1. $P4_{16}$ Architectures

We summarize the concept of $P4_{16}$ architectures, describe externs, and give an overview of the most common $P4_{16}$ architectures.

4.1.1. Concept

As described before, $P4_{16}$ introduces the concept of P4 architectures as an intermediate layer between the core P4 language and the targets. A P4 architecture serves as programming models that represents the capabilities and the logical view of a target's P4 processing pipeline. P4 programs are developed for a specific P4 architecture. Such programs can be deployed on all targets that implement the same P4 architecture. The manufacturers of P4 targets provide P4 compilers that compile architecture-specific P4 programs into target-specific configuration binaries.

4.1.2. Externs

P4 architectures may provide additional functionalities that are not part of the P4 language core. Examples are checksum or hash computation units, random number generators, packet and byte counters, meters, registers, and many others. To make such extern functionalities usable, $P4_{16}$ introduces so-called *externs*.

Most of the externs have to be explicitly instantiated in P4 programs using their constructor method. The other methods provided by these externs can then be invoked on the given extern instance. Other externs (extern functions) do not require explicit instantiating.

Along with tables and value sets, P4 externs are allowed to preserve additional state between packets. That state may be accessible by the control plane, the data plane, or both. For example, the counter extern would preserve the number of packets or bytes that has been counted so that each new packet can properly increment it. The specifics of the state depend on the nature of the extern and cannot be specified in the language; this is done inside the vendor-specific API definitions.

While the P4 processing pipeline only allows packet header manipulation, extern functions may operate on packet payload as well.

4.1.3. Overview of Common $P4_{16}$ Architectures

We describe the four most common $P4_{16}$ architectures.

v1model. The v1model mimics the processing pipeline of $P4_{14}$. As depicted in Figure 16, it consists of a programmable parser, an ingress match action pipeline, a traffic manager, an egress match-action pipeline, and a deparser. It enables developers to convert $P4_{14}$ programs into $P4_{16}$ programs. Additional functionalities tracking the development of the reference P4 software switch Behavioral Model version 2 (bmv2) (see Section 5) are continuously added. All P4 examples in this paper are written using v1model.

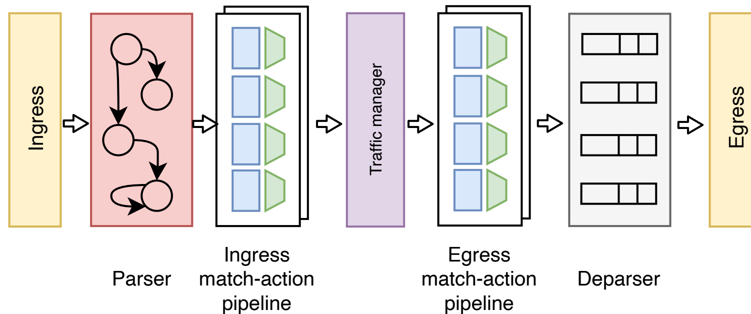


Figure 16: *v1model* architecture with a programmable parser, programmable ingress and egress match-action pipelines with a traffic manager in between, and a programmable parser.

Portable Switch Architecture (PSA). The PSA is a P4 architecture created and further developed by the Architecture WG [39] in the P4 Language Consortium. Besides, the WG also discusses standard functionalities, APIs, and externs that every target mapping the PSA should support. Its last specification is Version 1.1 [40] from November 2018. Figure 17 illustrates the P4 processing pipeline of the PSA. It is divided into an ingress and egress pipeline. Each pipeline consists of the three programmable parts: parser, multiple control blocks, and deparser. The architecture also defines configurable fixed-function components.

PSA specifies several packet processing primitives, such as:

- Sending a packet to an unicast port
- Dropping a packet
- Sending the packet to a multicast group
- Resubmitting a packet, which moves the currently processed packet from the end of the ingress pipeline to the beginning of the ingress pipeline for the purpose of packet re-parsing
- Recirculating a packet, which moves the currently processed packet from the end of the egress pipeline to the beginning of the ingress pipeline for the purposes of recursive processing, e.g., tunneling
- Cloning a packet, which duplicates the currently processed packet. *Clone ingress to egress (CI2E)* creates a duplicate of the ingress packet at the end of the ingress pipeline. *Clone egress to egress (CE2E)* creates a duplicate of the deparsed packet at the end of the egress pipeline. In both cases, cloned instances start processing at the beginning of the egress pipeline. Cloning can be helpful to implement powerful applications such as mirroring and telemetry.

SimpleSumeArchitecture. The SimpleSumeArchitecture is a simplified P4 architecture that is implemented by FPGA-based P4 targets. As depicted in Figure 18, it features a parser, a programmable match-and-action pipeline, and a deparser.

Tofino Native Architecture (TNA). TNA is a proprietary $P4_{16}$ architecture designed for Intel Tofino switching ASICs (see Section 5.3). Intel has published the architecture definitions and allows developers to publish programs written by using it.

The architecture describes a very high-performance, “industry-strength” device that is relatively complex. The basic programming unit is a so-called `Pipeline()` package that resembles an extended version of the Portable Switch Architecture (PSA) pipeline and consists of 6 top-level programmable components: the ingress parser, ingress match-action control, ingress deparser, and their egress counterparts. Since Tofino devices can have two or four processing

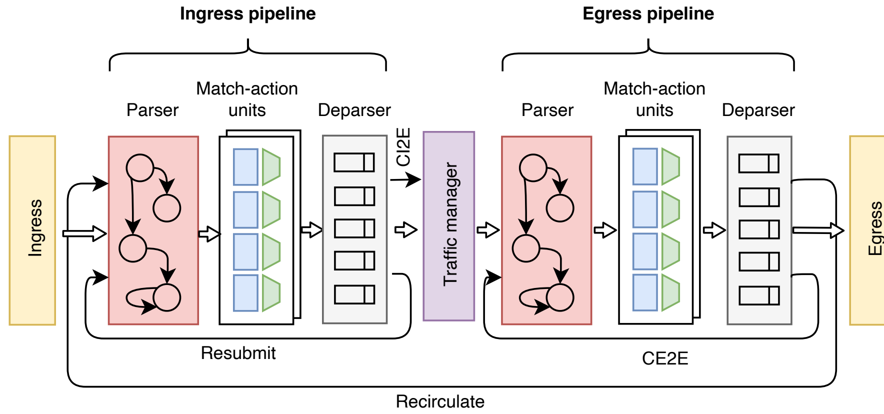


Figure 17: *Portable Switch Architecture (PSA)* with an ingress and egress pipeline and a traffic manager in between. Both include a programmable parser, programmable match-action units, a programmable deparser, fixed-function parts, and special packet processing primitives.

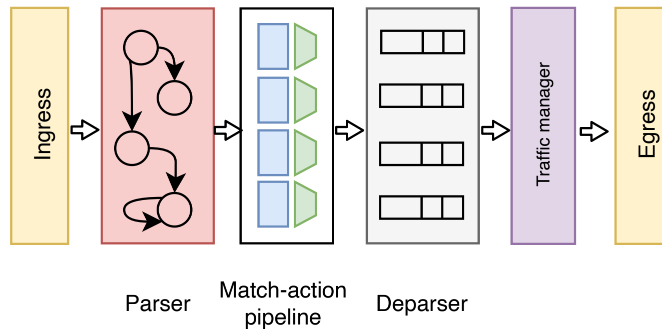


Figure 18: *SimpleSumeArchitecture* with a programmable parser, a programmable match-action pipeline, and a programmable parser followed by a traffic manager.

pipelines, the final switch package can be formed anywhere from one to four distinct pipeline packages. More complex versions of the `Pipeline()` package allow the programmer to specify different parsers for different ports.

TNA also provides a richer set of externs compared to most other architectures. Most notable is TNA `RegisterAction()` which represents a small code fragment that can be executed on the register instead of simple read/write operations provided in other architectures. TNA provides a clear and consistent interface for mirroring and resubmit with additional metadata being passed via the packet byte stream. The same technique is also used to pass intrinsic metadata which greatly simplifies the design.

Additional externs that are not present in other architectures include low-pass filters, weighted random early discard externs, powerful hash externs that can compute CRC based on user-defined polynomials, ParserCounter, and oth-

ers.

The set of intrinsic metadata in Tofino is also larger than in most other P4 architectures as presented before. Notable is support for two-level multicasting with additional source pruning, copy-to-cpu functionality, and support for IEEE 1588.

4.2. P4 Compiler

P4 compilers translate P4 programs into target-specific configuration binaries that can be executed on P4 targets. We first explain compilers based on the two-layer model which are most widely in use. Then we mention other compilers in less detail.

4.2.1. Two-Layer Compiler Model

Most P4 compilers use the two-layer model, consisting of a common frontend and a target-specific backend.

The frontend is common for all the targets and is responsible for parsing, syntactic and target-independent semantic analysis of the program. The program is finally transformed into an intermediate representation (IR) that is then consumed by the target-specific backend which performs target-specific transformations.

The first-generation P4 compiler for $P4_{14}$ was written in Python and used the so-called high-level intermediate representation (HLIR) [41] that represented $P4_{14}$ program as a tree of Python objects. The compiler is referred to as p4-hlir.

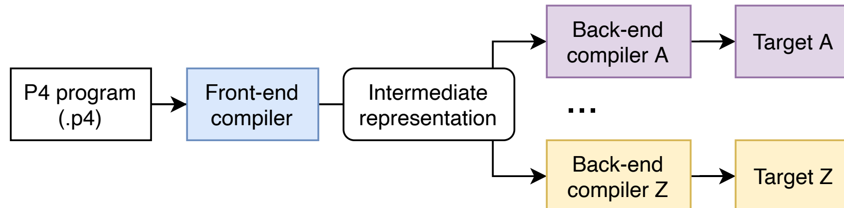


Figure 19: Structure and operation principle of P4 compilers using the two-layer model. The front-end compiler translates the given P4 program into an intermediate representation that is then compiled into target-specific code by back-end compilers.

The new P4 compiler (p4c) [42] is written in C++ and uses C++-object-based IR. As an additional benefit, the IR can be output as a $P4_{16}$ program or a JSON file. The latter allows the developers and users to build powerful tools for program analysis without the need to augment the compiler. Figure 19 visualizes its structure and operating principle. The compiler consists of a generic frontend that accepts both $P4_{14}$ and $P4_{16}$ code which may be written for any architecture. It furthermore has several reference backends for the bmv2, eBPF, and uBPF P4 targets as well as a backend for testing purposes and a backend that can generate graphs of control flows of P4 programs. In addition, p4c provides the so-called “mid-end” which is a library of generic transformation passes that are

used by the reference backends and can also be used by vendor-specific backends. The compiler is developed and maintained by P4.org.

P4 target vendors design and maintain their own compilers that include the common frontend. This ensures the uniformity of the language which is accepted by different compilers.

4.2.2. Other Compilers

MACSAD [43] is a compiler that translates P4 programs into Open Data Plane (ODP) [44] programs. Jose et al. [45] introduce a compiler that maps P4 programs to FlexPipe and RMT, two common software switch architectures. P4GPU [46] is a multistage framework that translates a P4 program into intermediate representations and other languages to eventually generate GPU code.

5. P4 Targets

We describe P4 targets based on software, FPGA, ASIC, and NPU. Table 3 compiles an overview of the targets, their supported architectures, and the current state of development.

5.1. Software-Based P4 Targets

Software-based P4 targets are packet forwarding programs that run on a standard CPU. We describe the 9 software-based P4 targets mentioned in Table 3.

5.1.1. p4c-behavioural

p4c-behavioral [47] is a combined P4 compiler and P4 software target. It was introduced with the first public release of P4. p4c-behavioral translates the given $P4_{14}$ program into an executable C program.

5.1.2. Behavioral Model version 2 (bmv2)

The second version of the P4 software switch Behavioral Model (bmv2) [48] was introduced to address the limitations of p4c-behavioural (see also [49]). In contrast to p4c-behavioral, the source code of bmv2 is static and independent of P4 programs. P4 programs are compiled to a JSON representation that is loaded onto the bmv2 during runtime. External functions and other extensions can be added by extending bmv2's C++ source code. bmv2 is not a single target, but a collection of targets [50]:

- *simple_switch* is the bmv2 target with the largest range of features. It contains all features from the $P4_{14}$ specification and supports the v1model architecture of $P4_{16}$. simple_switch includes a program-independent Thrift API for runtime control.
- *simple_switch_grpc* extends simple_switch by the P4Runtime API that is based on gRPC (see Section 6.3.1).

Table 3: Overview of P4 targets.

Target	P4 Version	$P4_{16}$ Architecture	Active Development
Software			
p4c-behavioral	$P4_{14}$	n.a.	X
bmv2	$P4_{14}$, $P4_{16}$	v1model, psa	✓
eBPF	$P4_{16}$	ebpf_model.p4	✓
uBPF	$P4_{16}$	ubpf_model.p4	✓
XDP	$P4_{16}$	xdp_model.p4	✓
T4P4S	$P4_{14}$, $P4_{16}$	v1model, psa	✓
Ripple	n.a	n.a	n.a
PISCES	$P4_{14}$	n.a.	X
PVPP	n.a.	n.a.	X
ZodiacFX	$P4_{16}$	zodiacfx_model.p4	n.a.
FPGA			
P4→NetFPGA	$P4_{16}$	SimpleSumeSwitch	✓
Netcope P4	n.a.	n.a.	✓
P4FPGA	$P4_{14}$, $P4_{16}$	n.a.	X
ASIC			
Barefoot Tofino/Tofino 2	$P4_{14}$, $P4_{16}$	v1model, psa, TNA	✓
Pensando Capri	$P4_{16}$	n.a	✓
NPU			
Netronome	$P4_{14}$, $P4_{16}$	v1model	✓

- *psa_switch* is similar to simple_switch, but supports PSA instead of v1model.
- *simple_router* and *l2_switch* support only parts of the standard metadata and do not support $P4_{16}$. They are intended to show how different architectures can be implemented with bmv2.

Although bmv2 is intended for testing purposes only, throughput rates up to 1 Gbit/s for a P4 program with IPv4 LPM routing have been reported [51]. bmv2 is under active development, i.e., new functionality is added frequently.

5.1.3. BPF-based Targets

Berkeley Packet Filters (BPFs) add an interface on a UNIX system that allows sending and receiving raw packets via the data link layer. User space programs may rely on BPFs to filter packets that are sent to it. BPF-based P4 targets are mostly intended for programming packet filters or basic forwarding in P4.

eBPF. Extended Berkeley Packet Filters (eBPFs) are an extension of BPFs for the Linux kernel. eBPF programs are dynamically loaded into the Linux kernel and executed in a virtual machine (VM). They can be linked to functions in the kernel, inserted into the network data path via iproute2, or bound to sockets or network interfaces. eBPF programs are always verified by the kernel before execution, e.g., programs with loops or backward pointers would not be executed. Due to their execution in a VM, eBPF programs can only access certain regions in memory besides the local stack. Accessing kernel resources is protected by a white list. eBPF programs may not block and sleep, and usage of locks is limited to prevent deadlocks. The p4c compiler features the *p4c-ebpf* back-end to compile $P4_{16}$ programs to eBPF [52].

uBPF. user-space BPFs (uBPFs) relocate the eBPF VM from the kernel space to the user space. *p4c-ubpf* [53] is a backend for p4c that compiles P4 HLIR for uBPF. In contrast to p4c-ebpf, it also supports packet modification, checksum calculation, and registers, but no counters.

XDP. eXpress Data Path (XDP) is based on eBPF and allows to load an eBPF program into the RX queue of a device driver. p4c-xdp [54] is a backend for p4c that compiles P4 HLIR for XDP. Similar to p4c-ubpf, it supports packet modification and checksum calculation. In contrast to p4c-ebpf, it supports counters instead of registers.

5.1.4. T_4P_4S

T_4P_4S (pronounced "tapas") [55, 56] is a software P4 target that relies on interfaces for accelerated packet processing such as Data Plane Development Kit (DPDK) [57] or Open Data Plane (ODP) [44]. T_4P_4S provides a compiler that translates P4 programs into target-independent C code that interfaces a network hardware abstraction library. Hardware-dependent and hardware-independent functionalities are separated from each other. Its source code is available on GitHub [58]. Bhardwaj et al. [59] describe optimizations for improving T_4P_4S performance by up to 15%.

5.1.5. Ripple

Ripple [60] is a P4 target based on DPDK. It uses a static universal binary that is independent of the P4 program. The data plane of the static binary is configured at runtime based on P4 HLIR. This results in a shorter downtime when updating a P4 program in contrast to targets like T_4P_4S. Ripple uses vectorization to increase the performance of packet processing.

5.1.6. PISCES

PISCES [61] transforms the Open vSwitch (OVS) [62] into a software P4 target. OVS is a popular SDN software switch that is designed for high throughput on virtualization platforms for flexible networking between VMs. The PISCES compiler translates P4 programs into C code that replace parts of the source code of OVS. This makes OVS dependent on the P4 program, i.e., OVS must

be recompiled with every modification of the P4 program. PISCES does not support stateful components such as registers, counters, or meters. The developers claim that PISCES does not add performance overhead to OVS. As the last commit in the public repository [63] is from 2016, PISCES seems not to be under active development.

5.1.7. PVPP

PVPP [64, 65] integrates P4 programs into plugins for Vector Packet Processors (VPP) (see Section 2.4.1). The P4-to-PVPP compiler comprises two stages. First, a modified p4c compiler translates P4 programs into target-dependent JSON code. Then, a Python compiler translates the JSON code into a VPP plugin in C source code. According to the authors, performance decreases by 5-17% compared to VPP but is still significantly better than OVS. Unfortunately, the source code and further information are not available for the public.

5.1.8. ZodiacFX

The ZodiacFX is a lightweight development and experimentation board originally designed as OF switch featuring four Fast Ethernet ports. It is based on an Atmel processor and an Ethernet switching chip [66]. The authors provided an extension [67, 68] to run P4 programs on the board. P4 programs are compiled using an extended version of p4c and the p4c-zodiacfx backend compiler. Then, the result of this compilation is used to generate a firmware image. Zanna et al. [69] compare the performance of P4 and OF on that target, and find out that differences among all test cases are small.

5.2. FPGA-Based P4 Targets

Several tool chains translate P4 programs into implementations for field programmable gate arrays (FPGAs). The process includes logic synthesis, verification, validation, and placement/routing of the logic circuit for the FPGA. We describe the P4→NetFPGA, Netcope P4, and P4FPGA tool chain. Finally, we mention research results for FPGA-based P4 targets.

5.2.1. P4→NetFPGA

The P4→NetFPGA workflow [70, 71] provides a development environment for compiling and running P4 programs on the NetFPGA SUME board that provides four SFP+ ports [72]. The development environment is built around the P4-SDnet compiler and the SDnet data plane builder from Xilinx, i.e., a full license for the Xilinx Vivado design suite is needed. Custom external functions can be implemented in a hardware description language (HDL) such as Verilog and included in the final FPGA program. This also allows external IP cores to be integrated as P4 externs in P4 programs. The P4→NetFPGA tool chain supports $P4_{16}$ based on the P4 architecture SimpleSumeSwitch (see Section 4.1).

5.2.2. Netcope P4

Netcope P4 [73] is a commercial cloud service that creates FPGA firmware from P4 programs. Knowledge of HDL development is not needed and all necessary IP cores are provided by Netcope. The cloud service can be used in conjunction with the Netcope software development kit (SDK). This combination allows developers to combine the VHDL code of the cloud service with custom HDL code, e.g., from an external function. As target platform, Netcope P4 supports FPGA boards from Netcope, Silicom, and Intel that are based on Xilinx or Intel FPGAs.

5.2.3. P4FPGA

P4FPGA [74] is a $P4_{14}$ and $P4_{16}$ compiler and runtime for the Bluespec programming language that can generate code for Xilinx and Altera FPGAs. The last commit in the archived public repository [75] is from 2017.

5.2.4. Research Results

Benácek and Kubátová [76, 77] present how P4 parse graph descriptions can be converted to optimized VHDL code for FPGAs. The authors demonstrate how a complex parser for several header fields achieves a throughput of 100 Gbit/s on a Xilinx Virtex-7 FPGA while using 2.78% slice look up tables (LUTs) and 0.76% slice registers (REGs). In a follow-up work [78], the optimized parser architecture supports a throughput of 1 Tbit/s on Xilinx UltraScale+ FPGAs and 800 Gbit/s on Xilinx Virtex-7 FPGAs. Da Silva et al. [79] also investigate the high-level synthesis of packet parsers in FPGAs. Kekely and Korenek [80] describe how MATs can be mapped to FPGAs. Iša et al. [81] describe a system for automated verification of register-transfer level (RTL) generated from P4 source code. Cao et al. [82, 83] propose a template-based process to convert P4 programs to VHDL. They use a standard P4 frontend compiler to compile the P4 program into an intermediate representation. From this representation, a custom compiler maps the different elements of the P4 program to VHDL templates which are used to generate the FPGA code.

5.3. ASIC-Based P4 Targets

5.3.1. Intel Tofino

Intel Tofino is the world's first user programmable Ethernet switch ASIC. It is designed for very high throughput of 6.5 Tbit/s (4.88 B pps) with 65 ports running at 100 Gbit/s. Its successor, the Tofino 2 ASIC, supports throughput rates of up to 12.8 Tbit/s with ports running at up to 400 Gbit/s. Tofino has been built by Barefoot Networks, a former startup company that was acquired by Intel in 2019.

The Tofino ASIC implements the TNA, a custom P4 architecture that significantly extends PSA (see Section 4.1). It provides support for advanced device capabilities which are required to implement complex, industrial-strength data plane programs. The device comes with 2 or 4 independent packet processing pipelines (pipes), each capable of serving 16 100 Gbit/s ports. All pipes can

run the same P4 program or each pipe can run its own program independently. Pipes can also be connected together, allowing the programmers to build programs requiring longer processing pipelines.

The Tofino ASIC processes packets at line rate irrespective of the complexity of the executed P4 program. This is achieved by a high degree of pipelining (each pipe is capable of processing hundreds of packets simultaneously) and parallelization. In addition to standard arithmetic and logical operations, Tofino provides specialized capabilities, often required by data plane programs, such as hash computation units and random number generators. For stateful processing Tofino offers counters, meters, and registers, as well as more specialized processing units. Some of them support specialized operations, such as approximate non-linear computations required to implement state-of-the-art data plane algorithms. Built-in packet generators allow the data plane designers to implement protocols, such as BFD, without using externally running control plane processes. These and other components are exposed through TNA which is openly published by Intel [84].

Tofino fixed-function components offer plenty of advanced functionality. The buffering engine has a unified 22 MB buffer, shared by all the pipes, that can be subdivided into several pools. Tofino Traffic Manager supports both store-and-forward as well as the cut-through mode, up to 32 queues per port, precise traffic shaping and multiple scheduling disciplines. Tofino provides nanosecond-precision timestamping that facilitates both the implementation of time synchronization protocols, such as IEEE 1588, as well as precise delay measurements. Additional intrinsic metadata support a variety of telemetry applications, such as INT.

The development is conducted using Intel P4 Studio which is a software development environment containing the P4 compiler, the driver, and other software necessary to program and manage the Tofino. A special interactive visualization tool (P4i) allows the developers to see the P4 program being mapped onto the specific hardware resources further assisting them in fitting and optimizing their programs. Intel P4 compiler for Tofino has special capabilities, allowing it to parallelize the code thereby taking advantage of the highly parallel nature of Tofino hardware.

A number of original design manufacturers (ODMs) produce open systems (white boxes) with the Tofino ASIC that are used for research, development, and production of custom systems. Examples include the EdgeCore Wedge 100BF-32X [85], APS Networks BF2556-1T-A1F [86] and BF6064-T-A2F [87], NetBerg Aurora 610 [88], and others.

Most white box systems follow a modern, server-like design with a separate board management controller, responsible for handling power supplies, fans, LEDs, etc., and a main CPU, typically x86_64, running a Linux operating system. The main CPU is connected to the Tofino ASIC via a PCIe interface. Some boards also provide one or more high-speed on-board Ethernet connections for faster packet interface. External Ethernet ports support speeds from 10 Gbit/s to 100 Gbit/s using standard QSFP28 cages although some systems offer lower-speed (1 Gbit/s) ports as well. Most of these systems are also powerful enough

to support running development tools natively, e.g., a P4 compiler, even though this is not necessarily required.

Tofino ASICs are also used in proprietary network switches, e.g., by Arista [89] and Cisco [90]. Some Tofino-based switches are supported by Microsoft SONiC [91].

5.3.2. Pensando Capri

The Capri P4 Programmable Processor [92, 93] is an ASIC that powers network interface cards (NICs) by Pensando Systems aimed for cloud providers. It is coupled with fixed function components for cryptography operations like AES or compression algorithms and features multiple ARM cores.

5.4. NPU-Based P4 Targets

Network processing units (NPUs) are software-programmable ASICs that are optimized for networking applications. They are part of standalone network devices or device boards, e.g., PCI cards.

Netronome network flow processing (NFP) silicons can be programmed with P4 [94] or C [95]. A C-based programming model is available that supports program functions to access payloads and allows developing P4 externs. The Agilio P4C SDK consists of a tool chain including a backend compiler, host software, and a full-featured integrated development environment (IDE). All current Agilio SmartNICs based on NFP-4000, NFP-5000, and NFP-6480 are supported. Harkous et al. [96] investigate the impact of basic P4 constructs on packet latency on Agilio SmartNICs.

6. P4 Data Plane APIs

We introduce data plane APIs for P4, present a characterization, describe the three most commonly used P4 data plane APIs, and compare different control plane use cases.

6.1. Definition & Functionality

Control planes manage the runtime behavior of P4 targets via data plane APIs. Alternative terms are *control plane APIs* and *runtime APIs*. The data plane API is provided by a device driver or an equivalent software component. It exposes data plane features to the control plane in a well-defined way. Figure 20 shows the main control plane operations. Most important, data plane APIs facilitate runtime control of P4 entities (MATs and externs). They typically also comprise a packet I/O mechanism to stream packets to/from the control plane. They also include reconfiguration mechanisms to load P4 programs onto the P4 target. Control planes can control data planes only through data plane APIs, i.e., if a data plane feature is not exposed via a corresponding API, it cannot be used by the control plane.

It is important to note that P4 does not require a data plane APIs. P4 targets may also be used as a packet processor with a fixed behavior that is defined by the P4 program where static MAT entries are part of the P4 program itself.

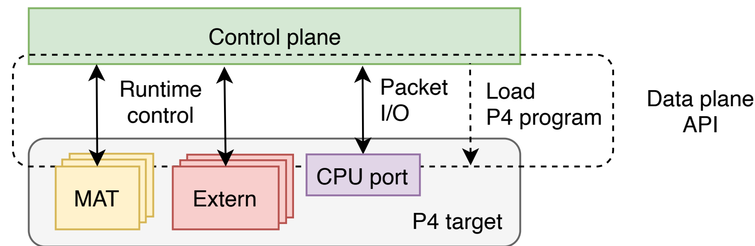


Figure 20: Runtime management of a P4 target by the control plane through the data plane API. The figure depicts the four most central operations: Runtime control of MATs and extern objects, packet-in/out, and loading of P4 programs.

6.2. Characterization of Data Plane APIs

Data plane APIs in P4 can be characterized by their level of abstraction, their dependency on the P4 program, and the location of the control plane.

6.2.1. Level of Abstraction

Data plane APIs can be characterized by their level of abstraction.

- *Device access APIs* provide direct access to hardware functionalities like device registers or memories. They typically use low-level mechanisms like DMA transactions. While this results in very low overhead, this type of API can be neither vendor- nor device-independent.

- *Data plane specific APIs* are APIs with a higher level of abstraction. They provide access to objects defined by the P4 program instead of hardware-specific parts. In contrast to device access APIs, vendor- and device-independence is possible for this type of API.

6.2.2. Dependency on the P4 Program

Data plane APIs can be characterized by their dependency on the P4 program.

- *Program-dependent APIs* have a set of functions, data structures, and other names that are derived from the P4 program itself. Therefore, they depend on the P4 program and are applicable to this P4 program only. If the corresponding P4 program is changed, function names, data structures, etc., might change, which requires a recompilation or modification of the control plane program.

- *Program-independent APIs* consist of a fixed set of functions that receives a list of P4 objects that are defined in the P4 program. Thus, the names of the API functions, data structures, etc., do not depend on the program and are universally applicable. If the corresponding P4 program changes, neither the names, nor the definitions of the API functions will change

as long as the control plane "knows" the names of the right tables, fields and other object that need to be operated on. Program-independent APIs model configurable objects either with the *object-based* or the *table-based* approach. As known from object-oriented programming, the object-based approach relies on methods that are defined for each class of data plane objects. In contrast, the table-based approach treats every class of data plane object as a variation of a table. This reduces the number of API methods as only table manipulations need to be provided as methods.

6.2.3. Control Plane Location

Data plane APIs can be characterized by the location of the control plane.

- *APIs for local control* are implemented by the device driver and are executed on the local CPU of the device that hosts the programmable data plane. Usually, the APIs are presented as set of C function calls just like for other devices that operating system are accessing.
- *APIs for remote control* add the ability to invoke API calls from a separate system. This increases system stability and modularity, and is essential for SDN and other systems with centralized control. Remote control APIs follow the base methodology of remote procedure calls (RPCs) but rely on modern message-based frameworks that allow asynchronous communication and concurrent calls to the API. Examples are Thrift [97] or gRPC [98]. For example, gRPC uses HTTP/2 for transport and includes many functionalities ranging from access authentication, streaming, and flow control. The protocol's data structures, services, and serialization schemes are described with protocol buffers (protobuf) [99].

6.3. Data Plane API Implementations

We introduce the three most common data plane APIs: P4Runtime, Barefoot Runtime Interface (BRI), and BM Runtime. All of them are data-plane specific and program-independent. Table 4 lists their properties that have been introduced before.

6.3.1. P4Runtime API

P4Runtime is one of the most commonly used data plane APIs that is standardized in the API WG [100] of the P4 Language Consortium. For implementing the RPC mechanisms, it relies on the gRPC framework with protobuf. Its most recent specification v1.3.0 [101] was published in December 2020.

Operating Principle. Figure 21 depicts the operating principle of P4Runtime. P4 targets include a gRPC server, controllers implement a gRPC client. To protect the gRPC connection, TLS with optional mutual certificate authentication can be enabled. The API structure of P4Runtime is described within the `p4runtime.proto` definition. The gRPC server on P4 targets interacts with the P4-programmable components via platform drivers. It has access to

P4 entities (MATs or externs) and can load target-specific configuration binaries. The structure of the API calls to access P4 entities are described in the `p4info.proto`. It is part of the P4Runtime but developers can extend it to use custom data structures, e.g., to implement interaction with target-specific externs. P4Runtime provides support for multiple controllers. For every P4 entity, read access is provided to all controllers whereas write access is only provided to one controller. To manage this access, P4 entities can be arranged in groups where each group is assigned to one primary controller with write access and arbitrary, secondary controllers with read access. Interaction between controllers and P4 targets works as follows. P4 compilers (see Section 4.2) with support for P4Runtime generate a P4Runtime configuration. It consists of the target-specific configuration binaries and P4Info metadata. P4Info describes all P4 entities (MATs and externs) that can be accessed by controllers via P4Runtime. Then, the controllers establish a gRPC connection to the gRPC server on the P4 target. The target-specific configuration is loaded onto the P4 target and P4 entities can be accessed.

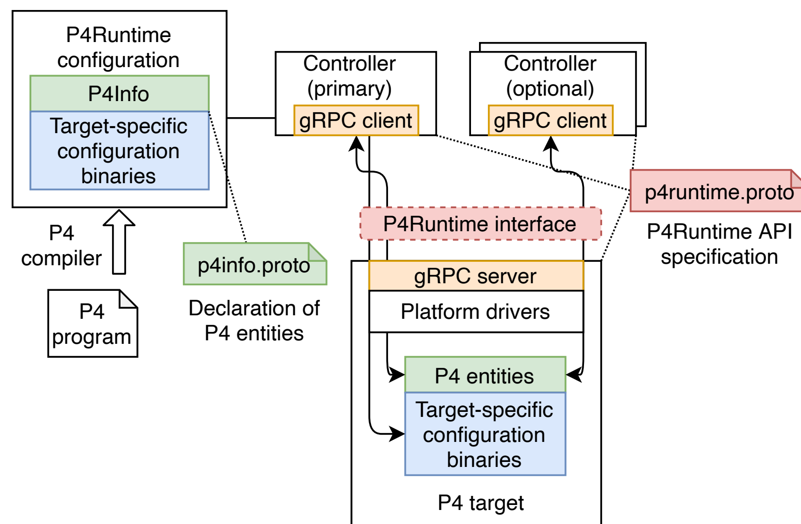


Figure 21: P4Runtime architecture (similar to [101]). P4 targets can be managed by a primary controller and multiple, optional controllers. The P4 entities and P4Runtime API specification is part of protocol definitions.

Implementations. gRPC and protobuf libraries are available for many high-level programming languages such as C++, Java, Go, or Python. Thereby, P4Runtime can be implemented easily on both controllers and P4 targets.

- *Controllers*: P4Runtime is supported by most common SDN controllers. P4 brigade [102] introduces support for P4Runtime on the Open Network

Operating System (ONOS). OpenDaylight (ODL) introduces support for P4Runtime via a plugin [103]. Stratum [104] is an open-source network operating system that includes an implementation of the P4Runtime and OpenConfig interfaces. Custom controllers, e.g., for P4 prototypes, can be implemented in Python with the help of the p4runtime_lib [105].

- *Targets*: The *PI Library* [106] is the open-source reference implementation of a P4Runtime gRPC server in C. It implements functionality for accessing MATs and supports extensions for target-specific configuration objects, e.g., registers of a hardware P4 target. The PI Library is used by many P4 targets including bmv2 [107] and the Tofino.

6.3.2. Barefoot Runtime Interface (BRI)

The BRI consists of two independent APIs that are available on Tofino-based P4 hardware targets. The *BfRt API* is an API for local control. It includes C, C++ and Python bindings that can be used to implement control plane programs. The *BF Runtime* is an API for remote control. As for P4Runtime, it is based on the gRPC RPC framework and protobuf, i.e., bindings for different languages are available. An additional Python library implements a simpler, BfRt-like interface for cases where simplicity is more essential than the performance of BF Runtime.

6.3.3. BM Runtime API

BM Runtime API is a program-independent data plane API for the bmv2 software target. It relies on the Thrift RPC framework. bmv2 includes a command line interface (CLI) program [108] to manipulate MATs and configure the multicast engine of the bmv2 P4 software target via this API.

Table 4: Characterization of data plane specific APIs.

API	**Program independence**	**Control plane location**
P4Runtime	✓	Remote (gRPC)
BF Runtime	✓	Remote (gRPC)
BfRt API	✓	Local (C, C++ and Python bindings)
BM Runtime	✓	Remote (Thrift RPC)

6.4. Controller Use Case Patterns

We present three use case patterns which are abstractions of the controller use cases introduced in the P4Runtime specification [101]. However, these are neither conclusive nor complete as derivations or extensions are possible.

6.4.1. Embedded/Local Controller

P4 hardware targets (see Section 5) comprise or are attached to a computing platform. This facilitates running controllers directly on the P4 target. Figure 22 depicts this setup. The controller application may either use a local API, e.g., C calls, or just execute a controller application that interfaces the data plane via an RPC channel.

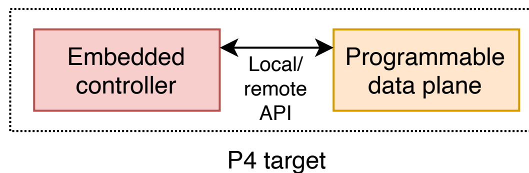


Figure 22: Embedded/local controller use case pattern. The P4 target comprises an embedded controller that is running a control plane program.

6.4.2. Remote Controllers

Remote controllers resemble the typical SDN setup where data plane devices are managed by a centralized control plane with an overall view on the network. Controllers need to be protected against outages and capacity overload, i.e., they need to be replicated for fail-safety and scalability. Figure 23 depicts two possible use cases. In the first shown use case (a), the programmable data plane on the P4 target is managed by remote controllers. In the second shown use case (b), the P4 target is managed by both, the embedded controller and remote controllers. Remote controllers might be interfaced using the remote API of the programmable data plane or an arbitrary API that is provided by the embedded controller. This option is often used for the implementation of so-called *hierarchical control plane* structures where control plane functionality is distributed among different layers. Control plane functions that do not require a global view of the network, e.g., link discovery, MAC learning for L2 forwarding, or port status monitoring, can be solely performed by the embedded/local controller. Other control plane functions that require an overall view of the network, e.g., routing applications, can be performed by the remote controller, possibly in cooperation with the embedded/local controller where the local controller acts as proxy, i.e., it relays control plane messages between the P4 target and the global controller. Hierarchical control planes improve load distribution as many tasks can be performed locally, which reduces load on the remote controllers. In particular, time-critical operations may benefit from local controllers as additional delays caused by the communication between a P4 target and a global controller are avoided.

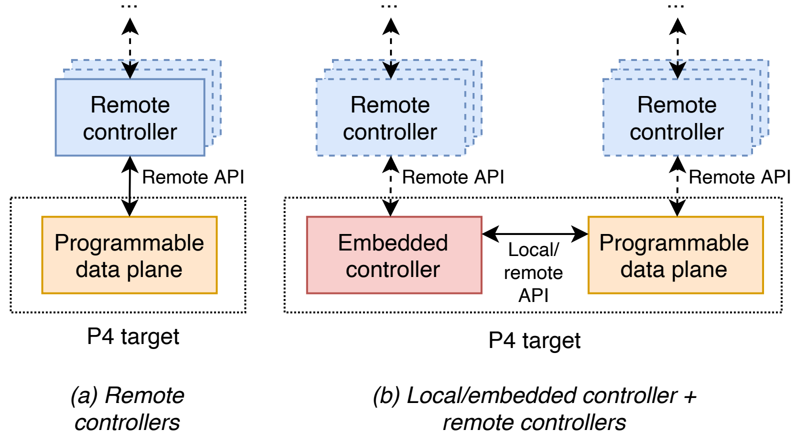


Figure 23: Two use case patterns for remote controllers: The P4 target may be solely managed by remote controllers or it may be managed by an embedded controller and remote controllers.

7. Advances in P4 Data Plane Programming

We give an overview on research to improve P4 data plane programming. Figure 24 depicts the structure of this section. We describe related work on optimization of development and deployment, testing and debugging, research on P4 targets, and research on control plane operation.

7.1. Optimization of Development and Deployment

We describe research work on optimizing the development & deployment process of P4.

7.1.1. Program Development

Graph-to-P4 [109] generates P4 program code for given parse graphs. This introduces a higher abstraction layer that is particularly helpful for beginners. Zhou et al. [110] introduce a module system for P4 to improve source code organization. DaPIPE [111] enables incremental deployment of P4 program code on P4 targets. SafeP4 [112] adds type safety to P4. P4I/O [113] presents a framework for intent-based networking with P4. Network operator describe their network functions with an Intent Definition Language (IDL) and P4I/O generates a complete P4 program accordingly. To that end, P4I/O provides a P4 action repository with various network functions. During reconfiguration, table and register state are preserved by applying backup mechanisms. P4I/O is implemented for a custom bmv2. Mantis [114] is a framework to implement fast reactions to changing network conditions in the data plane without controller interaction. To that end, annotations in the P4 code specify dynamic components and a quick control loop of those components ensure timely adjustments

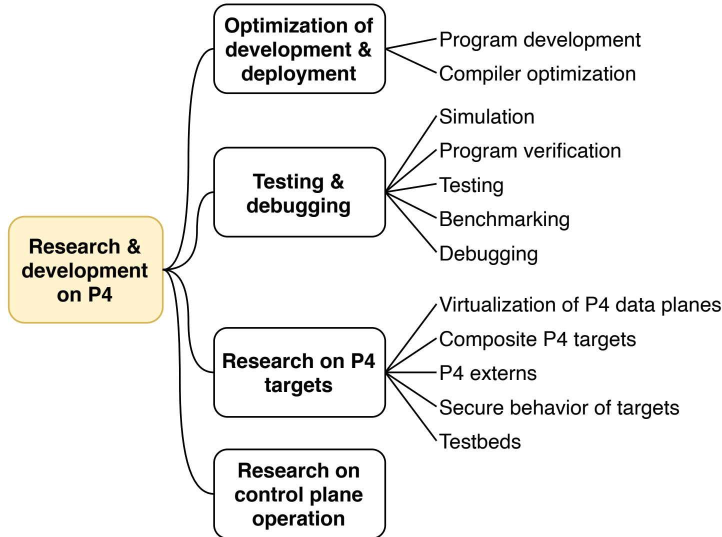


Figure 24: Organization of Section 7.

if necessary. Lyra [115] is a pipeline abstraction that allows developers to use simple statements to describe their desired data plane without low-level target-specific knowledge. Lyra then compiles that description to target-specific code for execution. GP4P4 [116] is a programming framework for self-driven networks. It generates P4 code from behavioral rules defined by the developer. To that end, GP4P4 evaluates the quality of the automatically generated programs and improves them based on genetic algorithms. FlowBlaze.p4 [117, 118, 119] implements an executor for FlowBlaze, an abstraction based on an extended finite state machine for building stateful packet processing functions, in P4. This library maps FlowBlaze elements to P4 components for execution on the bmv2. It also provides a GUI for defining the extended finite state machine. Flightplan [120] is a programming tool chain that disaggregates a P4 program into multiple P4 programs so that they can be executed on different targets. The authors state that this improves performance, resource utilization, and cost.

7.1.2. Compiler Optimization

pcube [121] is a preprocessor for P4 that translates primitive annotations in P4 programs into P4 code for common operations such as loops. CacheP4 [122] introduces a behavior-level cache in front of the P4 pipeline. It identifies flows and performs a compound of actions to avoid unnecessary table matches. The cache is filled during runtime by a controller that receives notifications from the switch. P5 [123] optimizes the P4 pipeline by removing inter-feature dependencies. dRMT [25] is a new architecture for programmable switches that introduces deterministic throughput and latency guarantees. Therefore, it gen-

erates schedules for CPU and memory resources from a P4 program. P2GO [124] leverages monitored traffic information to optimize resource allocation during compilation. It adjusts table and register size to reduce the pipeline length, and offloads rarely used parts of the program to the control plane. Yang et al. [125] propose a compiler module that optimizes lookup speed by reorganizing flow tables and prioritization of popular forwarding rules. Vass et al. [126] analyze and discuss algorithmic aspects of P4 compilation.

7.2. Testing and Debugging

We describe research work on simulation, program verification, testing, benchmarking, and debugging.

7.2.1. Simulation

PFPSim [127] is a simulator for validation of packet processing in P4. NS4 [128, 129] is a network simulator for P4 programs that is based on the network simulator NS3.

7.2.2. Program Verification

McKeown et al. [130] introduce a tool to translate P4 to the Datalog declarative programming language. Then, the Datalog representation of the P4 program can be analyzed for well-formedness. Kheradmand et al. [131] introduce a tool for static analysis of P4 programs that is based on formal semantics. P4v [132] adapts common verification methods for P4 that are based on annotations in the P4 program code. Freire et al. [133, 134] introduce assertion-based verification with symbolic execution. Stoenescu et al. [135] propose program verification based on symbolic execution in combination with a novel description language designed for the properties of P4. P4AIG [136] proposes to use hardware verification techniques where developers have to annotate their code with First Order Logic (FOL) specifications. P4AIG then encodes the P4 program as an Advanced-Inverter-Graph (AIG) which can be verified by hardware verification techniques such as circuit SAT solvers and bounded model checkers. bf4 [137] leverages static code verification and runtime checks of rules that are installed by the controller to confirm that the P4 program is running as intended. netdiff [138] uses symbolic execution to check if two data planes are equivalent. This can be useful to verify if a data plane behaves correctly by comparing it with a similar one, or to verify that optimizations of a data plane do not change its behavior. Yousefi et al. [139] present an abstraction for liveness verification of stateful network functions (NFs). The abstraction is based on boolean formulae. Further, they provide a compiler that translates these formulae into P4 programs.

7.2.3. Testing

P4pktgen [140] generates test cases for P4 programs by creating test packets and table entries. P4Tester [141] implements a detection scheme for runtime

faults in P4 programs based on probe packets. P4app [142] is a partially automated open source tool for building, running, debugging, and testing P4 programs with the help of Docker images. P4RL [143] is a reinforcement learning based system for testing P4 programs and P4 targets at runtime. The correct behavior is described in a simple query language so that a reinforcement agent based on Double DQN can learn how to manipulate and generate packets that contradict the expected behavior. P4TrafficTool [144] analyzes P4 programs to produce plugin code for common traffic analyzers and generators such as Wireshark.

7.2.4. Benchmarking

Whippersnapper [145] is a benchmark suite for P4 that differentiates between platform-independent and platform-specific tests. BB-Gen [146] is a system to evaluate P4 programs with existing benchmark tools by translating P4 code into other formats. P8 [147] estimates the average packet latency at compilation time by analyzing the data path program.

7.2.5. Debugging

Kodeswaran et al. [148] propose to use Ball-Larus encoding to track the packet execution path through a P4 program for more precise debugging capabilities. p4-data-flow [149] detects bugs by creating a control flow graph of a P4 program and then identifies incorrect behavior. P4box [150] extends the $P4_{16}$ reference compiler by so-called *monitors* that insert code before and after programmable blocks, e.g., control blocks, for runtime verification. P4DB [151] [152] introduces a runtime debugging system for P4 that leverages additional debugging snippets in the P4 program to generate reports during runtime. Neves et al. [153] propose a sandbox for P4 data plane programs for diagnosis and tracing. P4Consist [154] verifies the consistency between control and data plane. Therefore, it generates active probe-based traffic for which the control and data plane generate independent reports that can be compared later. KeySight [155] is a troubleshooting platform that analyzes network telemetry data for detecting runtime faults. Gauntlet [156] finds both crash bugs, i.e., abnormal termination of compilation operation, and semantic bugs, i.e., miscompilation, in compilers for programmable packet processors.

7.3. Research on P4 Targets

We describe research work on virtualization of P4 data planes, composite targets, P4 externs, secure behavior of targets, and testbeds.

7.3.1. Virtualization of P4 Data Planes

P4 targets are designed to execute one P4 program at any given time. Virtualization aims at sharing the resources of P4 targets for multiple P4 programs. Krude et al. [157] provide theoretical discussions on how ASIC- and FPGA-based P4 targets can be shared between different tenants and how P4 programs can be made hot-pluggable.

HyPer4 [158] introduces virtualization for P4 data planes. It supports scenarios such as network slicing, network snapshotting, and virtual networking. To that end, a compiler translates P4 programs into table entries that configure the HyPer4 *persona*, a P4 program that contains implementations of basic primitives. However, HyPer4 does not support stateful memory (registers, counters, meters), LPM, range match types, and arbitrary checksums. The authors describe an implementation for bmv2 and perform experiments that reveal 80 to 90% lower performance in comparison to native execution.

HyperV [159, 160, 161] is a hypervisor for P4 data planes with modular programmability. It allows isolation and dynamic management of network functions. The authors implemented a prototype for the bmv2 P4 target. In comparison to Hyper4, HyperV achieves a 2.5x performance advantage in terms of bandwidth and latency while reducing required resources by a factor of 4. HyperVDP [162] extends HyperV by an implementation of a dynamic controller that supports instantiating network functions in virtual data planes.

P4VBox [163], also published as VirtP4 [164], is a virtualization framework for the NetFPGA SUME P4 target. It allows executing virtual switch instances in parallel and also to hot-swap them. In contrast to HyPer4, HyperV and HyperVDP, P4VBox achieves virtualization by partially re-configuring the hardware.

P4Visor [165] merges multiple P4 programs. This is done by program overlap analysis and compiler optimization. Programming In-Network Modular Extensions (PRIME) [166] also allows combining several P4 programs to a single program and to steer packets through the specific control flows.

P4click [167] does not only merge multiple P4 programs, but also combines the corresponding control plane blocks. The purpose of P4click is to increase the use of data plane programmability. P4click is currently in an early stage of development.

The Multi Tenant Portable Switch Architecture (MTPSA) [168] is a P4 architecture that offers performance isolation, resource isolation, and security isolation in a switch for multiple tenants. MTPSA is based on the PSA. It combines a *Superuser* pipeline that acts as a hypervisor with multiple user pipelines. User pipelines may only perform specific actions depending on their privileges. MTPSA is implemented for bmv2 and NetFPGA-SUME [169].

Han et al. [170] provide an overview of virtualization in programmable data planes with a focus on P4. They classify virtualization schemes into hypervisor and compiler-based approaches, followed by a discussion of pros and cons of the different schemes. The aforementioned works on virtualization of P4 data planes are described and compared in detail.

7.3.2. Composite P4 Target

Da Silva et al. [171] introduce the idea of composite P4 targets. This tries to solve the problem of target-dependent support of features. The composed data plane appears as one P4 target; it is emulated by a P4 software target but relies on an FPGA and ASIC for packet processing.

eXtra Large Table (XLT) [172] introduces gigabyte-scale MATs by leveraging FPGA and DRAM capabilities. It comprises a P4-capable ASIC and multiple FPGAs with DDR4 DRAM. The P4-capable ASIC pre-constructs the match key field and sends it with the full packet to the FPGA. The FPGA sends back the original packet with the search results of the MAT lookup. The authors implement a DPDK based prototype for the T_4P_4S P4 software target.

HyMoS [173] is a hybrid software and hardware switch to support NFV applications. The authors create a switch by using P4-enabled Smart NICs as line cards and the PCIe interface of a computer as the switch fabric. P4 is used for packet switching between the NICs. Additional processing may be done using DPDK or applications running on a GPU.

7.3.3. P4 Externs

Laki et al. [174, 175] investigate asynchronous execution of externs. In contrast to common synchronous execution, other packets may be processed by the pipeline while the extern function is running. The authors implement and evaluate a prototype for T4P4S. Scholz et al. [176] propose that P4 targets should be extended by cryptographic hash functions that are required to build secure applications and protocols. The authors propose an extension of the PSA and discuss the PoC implementation for a CPU-, network processing unit (NPU)-, and FPGA-based P4 target. Da Silva et al. [177] investigate the implementation of complex operations as extensions to P4. The authors perform a case study on integrating the Robust Header Compression (ROHC) scheme and conclude that an implementation as extern function is superior to an implementation as a new native primitive.

7.3.4. Secure Behaviour of Targets

Gray et al. [178] demonstrate that hardware details of P4 targets influence their packet processing behavior. The authors demonstrate this by sending a special traffic pattern to a P4 firewall. It fills the cache of this target and results in a blocking behavior although the overall data rate is far below the capacity of the used P4 target. Dumitru et al. [179] investigate the exploitation of programming bugs in bmv2, P4-NetFPGA, and Tofino. The authors demonstrate attack scenarios by header field access on invalid headers, the creation of infinite loops and unintentionally processing of dropped packets in the P4 targets.

7.3.5. Testbeds

Large testbeds facilitate research and development on P4 programs. The i-4PEN (International P4 Experimental Networks) [180] is an international P4 testbed operated by a collaboration of network research institutions from the USA, Canada, and Taiwan. Chung et al.[181] describe how multi-tenancy is achieved in this testbed. The 2STiC testbed [182], a national testbed in the Netherlands comprising six sites with at least one Tofino-based P4 target, is connected to i-4PEN.

7.4. Research on Control Plane Operation

When new forwarding entries are computed by the controller, the data plane has to be updated. However, updating the targets has to be performed in a manner that prevents negative side effects. For example, microloops may occur if packets are forwarded according to new rules at some targets while at other devices old rules are used because updates have to arrive yet.

Sukapuram et al. [183, 184] introduce a timestamp in the packet header that contains the sending time of a packet. When switches receive a packet during an update period, they compare the timestamp of both the packet and the update to determine whether a packet has been sent before the update, and thus, old rules should be used for forwarding.

Liu et al. [185] introduce a mechanism where once a packet is matched against a specific forwarding rule, it cannot be matched downstream on a rule that is older. To that end, the packet header contains a timestamp field that records when the last applied forwarding rule has been updated. If the packet is matched against an older rule, the packet is dropped, otherwise the timestamp is updated and the packet is forwarded.

Ez-Segway [186] facilitates updating by including data plane devices in the update process. When a data plane device receives an update, it determines which of its neighbors is affected by the update as well, and forwards the update to that neighbor. This prevents loops and black holes.

TableVisor [187] is a transparent proxy-layer between the control plane and data plane. It provides an abstraction from heterogeneous data plane devices. This facilitates the configuration of data plane switches with different properties, e.g., forwarding table size.

Molero et al. [188] propose to offload tasks from the control plane to the data plane. They show that programmable data planes are able to run typical control plane operations like failure detection and notification, and connectivity retrieval. They discuss trade-offs, limitations and future research opportunities.

8. Applied Research Domains: Classification & Overview

In the following sections, we give an overview of applied research conducted with P4. In this section, we classify P4's core features that make it attractive for the implementation of data plane algorithms. We define research domains, visualize them in a compact way, and explain our method to review corresponding research papers in the subsequent sections. Finally, we delimit the scope of the surveyed literature.

8.1. Classification of P4's Core Features

We identify P4's core features for the implementation of prototypes. We classify them in the following to effectively reason about P4's usefulness for the surveyed research works.

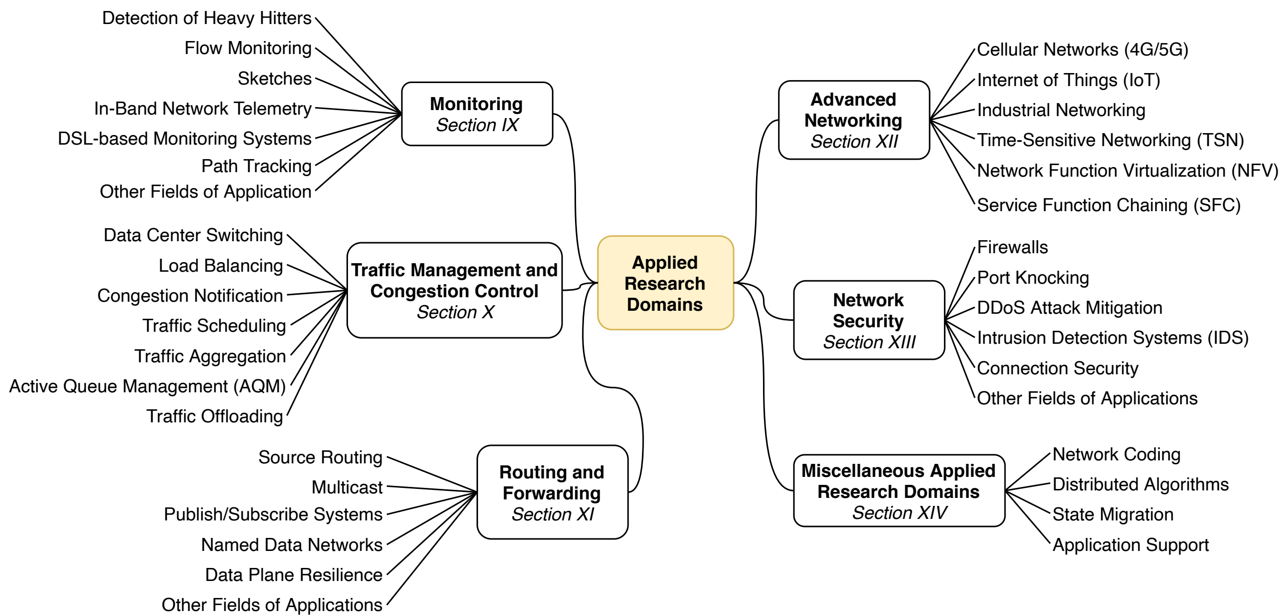


Figure 25: Categorization of the surveyed works into applied research domains and subdomains – they correspond to sections and subsections in the remainder of this paper.

8.1.1. Definition and Usage of Custom Packet Headers

P4 requires the definition of packet headers (Section 3.5). These may be headers of standard protocols, e.g., TCP, use-case-specific protocols, e.g., GTP in 5G, or new protocols. As P4 supports the definition of custom headers, it is suitable for the implementation of data plane algorithms using new protocols or extensions of existing protocols, e.g., for in-band signalling.

8.1.2. Flexible Packet Header Processing

Control blocks with MATs (Section 3.6) comprise the packet processing logic. Packet processing includes default actions, e.g., forwarding and header field modifications, or custom, user-defined actions. Both may be parameterized via MATs or metadata. Entries in the MATs are maintained by a data plane API (Section 6). The flexible use of actions, the definition of new actions, and their parameterization offer high flexibility for header processing, which is often needed for research prototypes.

8.1.3. Target-Specific Packet Header Processing Functions

While the above-mentioned features are part of the P4 core language and supported by any P4-capable platform, devices may offer additional architecture- or target-specific functionality which is made available as *P4 extern* (Section 4). Typical externs include components for stateful processing, e.g., registers or

counters, operations to resubmit/recirculate the packet in the data plane, multicast operations, or more complex operations, e.g., hashing and encryption/decryption. P4 software targets allow users to integrate custom externs and use them within P4 programs. While this is also possible to some extent on some P4 hardware targets, e.g., the NetFPGA SUME board, high-throughput P4 targets based on the Tofino ASIC have only a fixed set of externs (Section 5.3). Depending on the use case, the availability of externs may be essential for the implementation of prototypes. Thus, externs facilitate the implementation of more complex algorithms but make implementations platform-dependent.

8.1.4. Packet Processing on the Control Plane

Similar to control plane SDN (e.g., OF), more complex, and optionally centralized packet processing can be outsourced to an SDN control plane; packet exchange and data plane control is performed via a data plane API (Section 6). While OF only allows the exchange of complete packets, P4 enables the end-users to define the packet formats.

8.1.5. Flexible Development and Deployment

Users are able to easily change the P4 programs on P4 targets that are installed in a network. This facilitates agile development with frequent deployments and incremental functionality extensions by deploying new versions of a P4 programs.

8.2. Categorization of Research Domains

To organize the survey in the following sections, we define research domains and structure them in a two-level hierarchy as depicted in Figure 25. This categorization helps the reader to get a quick overview in certain applied areas and improves the readability of this survey. The choice of the research domains is dominated by the fields of applications, but the summaries of the sections will show that the prototypes in these areas benefit from different core features of P4.

For each research domain, we provide a table that lists the publications with publication year, P4 target platforms, and source code availability. This supports efficient browsing of the content and backs our conclusions in the section-specific summaries.

8.3. Scope of the Surveyed Literature

We consider the literature until the end of 2020 and selected papers from 2021, including journal papers, conference papers, workshop papers, and preprints. Out of the 377 scientific publications we surveyed in this work (see Section 1), 245 fall in the area of applied research. 68 of those research papers were published in 2018 or before, 80 were published in 2019, 93 were published in 2020, and 4 were published in 2021. 60 out of all 245 research publications released the source code of their prototype implementations.

Table 5 depicts a statistic on major publication venues for the papers of applied research domains. It helps the reader to identify potential venues for prospective own publications based on P4 technology.

9. Applied Research Domains: Monitoring

We describe applied research on detection of heavy hitters, flow monitoring, sketches, in-band network telemetry, and other areas of application. Table 6 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

9.1. Detection of Heavy Hitters

Heavy hitters [269] (or "elephant flows") are large traffic flows that are the major source of network congestion. Detection mechanisms aim at identifying heavy hitters to perform extra processing, e.g., queuing, flow rate control, and traffic engineering.

HashPipe [189] integrates a heavy hitter detection algorithm entirely on the P4 data plane. A pipeline of hash tables acts as a counter for detected flows. To fulfill memory constraints, the number of flows that can be stored is limited. When a new flow is detected, it replaces the flow with the lowest count. Thus, light flows are replaced, and heavy flows can be detected by a high count. Lin et al. [191] describe an enhanced version of the algorithm.

Popescu et al. [192] introduce a heavy hitter detection mechanism. The controller installs TCAM entries for specific source IP prefixes on the switch. If one of these entries matches more often than a threshold during a given time frame, the entry is split into two entries with a larger prefix size. This procedure is repeated until the configured granularity is reached.

Harrison et al. [193] presents a controller-based and distributed detection scheme for heavy hitters. The authors make use of counters for the match key values, e.g., source and destination IP pair or 5-tuple, that are maintained by P4 switches. If a counter exceeds a certain threshold, the P4 switch sends a notification to the controller. The controller generates more accurate status reports by combining the notifications received from the switches.

Kucera et al. [194] describe a system for detecting traffic aggregates. The authors propose a novel algorithm that supports hierarchical heavy hitter detection, change detection, and super-spreader detection. The complete mechanism is implemented on the P4 data plane and uses push notifications to a controller.

IDEAFIX [195] is a system that detects elephant flows at edge switches of Internet exchange point networks. The proposed system analyzes flow features, stores them with hash keys as indices in P4 registers, and compares them to thresholds for classification.

Turkovic et al. [196] propose a streaming approach for detecting heavy hitters via sliding windows that are implemented in P4. According to the authors, interval methods that are typically used to detect heavy hitters are not suitable

Table 5: Statistics of scientific publications regarding applied research conducted with P4.

Venue	#Publications
Journals	**41**
IEEE ACCESS	9
IEEE/ACM ToN	7
IEEE TNSM	6
JNCA	4
Miscellaneous	15
Conferences	**168**
ACM SOSR	14
IEEE NFV-SDN	12
IEEE ICNP	12
IEEE ICC	10
ACM SIGCOMM	10
IEEE/IFIP NOMS	8
ACM CoNEXT	7
IEEE NetSoft	7
USENIX NSDI	6
IEEE INFOCOM	6
ACM/IEEE ANCS	5
IFIP Networking	5
IEEE GLOBECOM	4
CNSM	4
IEEE CloudNet	3
APNOMS	3
IFIP/IEEE IM	3
Miscellaneous	49
Workshops	**36**
EuroP4	11
Morning Workshop on In-Network Computing	5
SPIN	3
ACM HotNets	3
INFOCOM Workshops	3
Miscellaneous	11

Table 6: Overview of applied research on monitoring (Section 9).

Research work	Year	Targets	Code
Detection of Heavy Hitters (Section 9.1)			
HashPipe [189]	2017	bmv2	[190]
Lin et al. [191]	2019	Tofino	
Popescu et al. [192]	2017	-	
Harrison et al. [193]	2018	Tofino	
Kucera et al. [194]	2020	bmv2	
IDEAFIX [195]	2018	-	
Turkovic et al. [196]	2019	Netronome	
Ding et al. [197]	2020	bmv2	[198]
Flow Monitoring (Section 9.2)			
TurboFlow [199]	2018	Tofino, Netronome	[200]
$*$Flow [201]	2018	Tofino	[202]
Hill et al. [203]	2018	bmv2	
FlowStalker [204]	2019	bmv2	
ShadowFS [205]	2020	bmv2	
FlowLens [206]	2021	bmv2, Tofino	[207]
SpiderMon [208]	2020	bmv2	
ConQuest [209]	2019	Tofino	
Zhao et al. [210]	2019	bmv2, Tofino	
Sketches (Section 9.3)			
SketchLearn [211]	2018	Tofino	[212]
MV-Sketch [213]	2020	bmv2, Tofino	[214]
Hang et al. [215]	2019	Tofino	
UnivMon [216]	2016	p4c-behavioural	
Yang et al. [217, 218]	2018/19	Tofino	[219]
Pereira et al. [220]	2017	bmv2	
Martins et al. [221]	2018	bmv2	
Lai et al. [222]	2019	Tofino	
Liu et al. [223]	2020	Tofino	
SpreadSketch [224]	2020	Tofino	[225]

Research work	Year	Targets	Code
In-Band Network Telemetry (Section 9.4)			
Vestin et al. [226]	2019	Netronome	
Wang et al. [227]	2019	Tofino	
IntOpt [228]	2019	P4FPGA	
Jia et al. [229]	2020	bmv2	[230]
Niu et al. [231]	2019	Tofino, Netronome	
CAPEST [232]	2020	bmv2	[233]
Choi et al. [234]	2019	bmv2	
Sgambelluri et al. [235]	2020	bmv2	
Feng et al. [236]	2020	Netronome	
IntSight [237]	2020	bmv2, NetFPGA-SUME	[238]
Suh et al. [239]	2020	-	
DSL-Based Monitoring Systems (Section 9.5)			
Marple [240, 241]	2017	bmv2	[242]
MAFIA [243]	2019	bmv2	[244]
Sonata [245]	2018	bmv2, Tofino	[246]
Teixeira et al. [247]	2020	bmv2, Tofino	
Path Tracking (Section 9.6)			
UniRope [248]	2018	bmv2, PISCES	
Knossen et al. [249]	2019	Netronome	
Basuki et al. [250]	2020	bmv2	
Other Areas of Application (Section 9.7)			
BurstRadar [251]	2018	Tofino	[252]
Dapper [253]	2017	-	
He et al. [254]	2018	Tofino	
Riesenberg et al. [255]	2019	bmv2	[256]
Wang et al. [257]	2020	Tofino	
P4STA [258]	2020	bmv2, Netronome	[259]
Hark et al. [260]	2019	-	
P4Entropy [261]	2020	bmv2	[262]
Taffet et al. [263]	2019	bmv2	
NetView [264]	2020	bmv2, Tofino	
FastFE [265]	2020	Tofino	
Unroller [266]	2020	bmv2, Netcope P4-to-VHDL	
Hang et al. [267]	2019	Tofino	
FlowSpy [268]	2019	bmv2	

for programmable data planes because of high hardware resources, bad accuracy, or a need for too much intervention by the control plane.

Ding et al. [197] propose an architecture for network-wide heavy hitter detection. The authors' main focuses are hybrid SDN/non-SDN networks where programmable devices are deployed only partially. To that end, they also present an algorithm for an incremental deployment of programmable devices with the goal of maximizing the number of network flows that can be monitored.

9.2. Flow Monitoring

In flow monitoring, traffic is analyzed on a per-flow level. Network devices are configured to export per-flow information, e.g., packet counters, source and target IP addresses, ports, or protocol types, as flow records to a flow collector. These flow records are often duplicates of network packets without payload data. The flow collector then performs centralized analysis on this data. The three most widely deployed protocols are Netflow [270], sFlow [271], and IPFIX [272].

TurboFlow [199] is a flow record generator designed for P4 switches that does not have to make use of sampling or mirroring. The data plane generates micro-flow records with information about the most recent packets of a flow. On the CPU module of the switch, those micro-flow records are aggregated and processed into full flow records.

"∗Flow" [201] partitions measurement queries between the data plane and a software component. A switching ASIC computes grouped packet vectors that contain a flow identifier and a variable set of packet features, e.g. packet size and timestamps, while the software component performs aggregation. "∗Flow" supports dynamic and concurrent measurement applications, i.e., measurement applications that operate on the same flows without impacting each other.

Hill et al. [203] implement Bloom filters on P4 switches to prevent sending duplicate flow samples. Bloom filters are a probabilistic data structure that can be used to check whether an entry is present in a set or not. It is possible to add elements to that set, but it is not possible to remove entries from it. For flow tracking, Bloom filters test if a flow has been seen before without control plane interaction. Thereby, only flow data is forwarded to the collector from flows that were not seen before.

FlowStalker [204] is a flow monitoring system running on the P4 data plane. The monitoring operations on a packet are divided in two phases, a proactive phase that identifies a flow and keeps a per-flow packet counter and a reactive phase that runs for large flows only and gathers metrics of the flow, e.g., byte counts and packet sizes. The controller gathers information from a cluster of switches by injecting a crawler packet that travels through the cluster at one switch. ShadowFS [205] extends FlowStalker with a mechanism to increase the throughput of the monitored flows. It achieves this by dividing forwarding tables into two tables, a faster and a slower one. The most utilized flows are moved to the faster table if necessary.

FlowLens [206] is a system for traffic classification to support security network applications based on machine learning algorithms. The authors propose

a novel memory-efficient representation for features of flows called *flow marker*. A profiler running in the control plane automatically generates an application-specific flow marker that optimizes the trade-off between resource consumption and classification accuracy, according to a given criterion selected by the operator.

SpiderMon [208] monitors network performance and debugs performance failures inside the network with little overhead. To that end, SpiderMon monitors every flow in the data plane and recognizes if the accumulated latency exceeds a certain threshold. Furthermore, SpiderMon is able to trace back the path of interfering flows, allowing to analyze the cause of the performance degradation.

ConQuest [209] is a data plane mechanism to identify flows that occupy large portions of buffers. Switches maintain snapshots of queues in registers to determine the contribution to queue occupancy of the flow of a received packet.

Zhao et al. [210] implement flow monitoring using hash tables. Using a novel strategy for collision resolution and record promotion, accurate records for elephant flows and summarized records for other flows are stored.

9.3. Sketches

Flow monitoring as described in Section 9.2 requires high sampling rates to produce sufficiently detailed data. As an alternative, streaming algorithms process sequential data streams and are subject to different constraints like limited memory or processing time per item. They approximate the current network status based on concluded summaries of the data stream. The streaming algorithms output so-called sketches that contain summarized information about selected properties of the last n packets of a flow.

SketchLearn [211] is a sketch-based approach to track the frequency of flow records. It features multilevel sketches that aim for small memory usage, fast per-packet processing, and real-time response. Rather than finding the perfect resource configuration for measurement traffic and regular traffic, SketchLearn characterizes the statistical error of resource conflicts based on Gaussian distributions. The learned properties are then used to increase the accuracy of the approximated measurements.

Tang et al. [213] present MV-Sketch, a fast and compact invertible sketch. MV-Sketch leverages the idea of majority voting to decide whether a flow is a heavy hitter or heavy changer. Evaluations show that MV-Sketch achieves a 3.38 times higher throughput than existing invertible sketches.

Hang et al. [215] try to solve the problem of inconsistency when a controller needs to collect the data from sketches on one or more switches. As accessing and clearing the sketches on the switches is always subject to latency, not all sketches are reset at the same time, and there might be some delay between accessing and clearing the sketches. The authors propose to use two asymmetric sketches on the switches that are used in an interleaved way. Furthermore, the authors propose to use a distributed control plane to keep latency low.

UnivMon [216] is a flow monitoring system based on sketches. After sampling the traffic, the data plane produces sketches and determines the top-k heaviest

flows by comparing the number of sketches for each flow. Those flows are passed to the control plane which processes the data for the specific application.

Yang et al. [217, 218] propose to adapt sketches according to certain traffic characteristics to increase data accuracy, e.g., during congestion or distributed denial of service (DDoS) attacks. The mechanism is based on compressing and merging sketches when resources in the network are limited due to high traffic volume. During periods with high packet rates, only the information of elephant flows is recorded to trade accuracy for higher processing speed.

Pereira et al. [220] propose a secured version of the Count-Min sketch. They replace the function with a cryptographic hash function and provide a way for secret key renewal.

Martins et al. [221] introduce sketches for multi-tenant environments. The authors implement bitmap and counter-array sketches using a new probabilistic data structure called BitMatrix that consists of multiple bitmaps that are stored in a single P4 register.

Lai et al. [222] use a sketch-based approach to estimate the entropy of network traffic. The authors use CRC32 hashes of header fields as match keys for match-action tables and subsequently update k-dimensional data sketches in registers. The content of the registers is then processed by the control plane CPU which calculates the entropy value.

Liu et al. [223] use sketches for performance monitoring. They introduce lean algorithms to measure metrics like loss or out-of-order packets.

SpreadSketch [224] is a sketch data structure to detect superspreaders. The sketch data structure is invertible, i.e., it is possible to extract the identification of superspreaders from the sketch at the end of an epoch.

9.4. In-Band Network Telemetry

Barefoot Networks, Arista, Dell, Intel and VMware specified in-band network telemetry (INT) specifically for P4 [273]. It uses a pure data plane implementation to collect telemetry data from the network without any intervention by the control plane. It was specified by INT is the main focus of the *Applications WG* [274] of the P4 Language Consortium. Instructions for INT-enabled devices that serve as traffic sources are embedded as header fields either into normal packets or into dedicated probe packets. Traffic sinks retrieve the results of instructions to traffic sources. In this way, traffic sinks have access to information about the data plane state of the INT-enabled devices that forwarded the packets containing the instructions for traffic sources. The authors of the INT specification name network troubleshooting, advanced congestion control, advanced routing, and network data plane verification as examples for high-level use cases.

In two demos, INT was used for diagnosing the cause of latency spikes during HTTP transfers [275] and for enforcing QoS policies on a per-packet basis across a metro network [276].

Vestin et al. [226] enhance INT traffic sinks by event detection. Instead of exporting telemetry items of all packets to a stream processor, exporting has to

be triggered by an event. Furthermore, they implement an INT report collector for Linux that can stream telemetry data to a Kafka cluster.

Wang et al. [227] design an INT system that can track which rules in MATs matched on a packet. The resulting data is stored in a database to facilitate visualization in a web UI.

IntOpt [228] uses INT to monitor service function chains. The system computes minimal monitoring flows that cover all desired telemetry demands, i.e., the number of INT-sources, sinks, and forwarding nodes that are covered by this flow is minimal. IntOpt uses active probing, i.e., monitoring probes for the monitoring flows are periodically inserted into the network.

Jia et al. [229] use INT to detect gray failures in data center networks using probe packets. Gray failures are failures that happen silently and without notification.

Niu et al. [231] design a multilevel INT system for IP-over-optical networks. Their goal is to monitor both the IP network and the optical network at the same time. To that end, they implement optical performance monitors for bandwidth-variable wavelength selective switches. Their measurements can be queried by a P4 switch that is connected directly to it.

CAPEST [232] leverages P4-enabled switches to estimate the network capacity and available bandwidth of network links. The approach is passive, i.e., it does not disturb the network. A controller sends INT probe packets to trigger statistical analysis and export results.

Choi et al. [234] leverage INT for run-time performance monitoring, verification, and healing of end-to-end services. P4-capable switches monitor the network based on INT information and the distributed control plane verifies that SLAs and other metrics are fulfilled. They leverage metric dynamic logic (MDL) to specify formal assertions for SLAs.

Sgambelluri et at. [235] propose a multi-layer monitoring system that uses an OpenConfig NETCONF agent for the optical layer an P4-based INT for the packet layer. In their prototype, they use INT to measure the delay of packets by computing the processing time at each switch.

Feng et al. [236] implement an INT sink for Netronome Smart NICs. After parsing the INT headers using P4, they use algorithms written in C to perform INT tasks like aggregation and notification. Compared to a pure P4 implementation, this increases the performance.

IntSight [237] is a system for detecting and analyzing violations of service-level objects (SLOs). SLOs are performance guarantees towards a network, e.g., concerning bandwidth and latency. IntSight uses INT to monitor the performance of the network during a specific period of time. Egress devices gather this information and produce a report at the end of the period if an SLO has been violated.

Suh et al. [239] explore how a sampling mechanism can be added to INT. Their solution supports rate-based and event-based sampling. Based on these sampling strategies, INT headers are only added to a fraction of the packets to reduce overhead.

9.5. DSL-Based Monitoring Systems

Monitoring tasks can often be broken down in a set of several basic operations, e.g., map, filter, or groupby. A domain-specific language (DSL) allows to combine these basic operations in more complex tasks.

Marple [240, 241] is a performance query language that supports existing constructs like map, filter, groupby, and zip. A query compiler translates the queries either to P4 or to a simulator for programmable switch hardware. Stateless constructs of the query language, e.g., filters, are executed on the data plane. Stateful constructs, e.g., groupby, use a programmable key-value store that is split between a fast on-chip SRAM cache and a large off-chip DRAM backing store. The results are streamed from the switch to a collection server.

MAFIA [243] is a DSL to describe network measurement tasks. They identify several fundamental primitive operations, examples are match, tag, timestamp, sketch, or counter. MAFIA is a high-level language to describe more complex measurement tasks composed of those primitives. The authors provide a Python-based compiler that translates MAFIA code into a P4 program in $P4_{14}$ or $P4_{16}$ for a PISA-based P4 target.

Sonata [245] is a query-driven telemetry system. It provides a query interface that provides common operators like map and reduce that can be applied on arbitrary packet fields. Sonata combines the capabilities of both programmable switches and stream processors. The queries are partitioned between the programmable switches and the stream processors to reduce the load on the stream processors. Teixeira et al. [247] extend the Sonata prototype by functionalities to monitor the properties of packet processing inside switches, e.g., delay.

9.6. Path Tracking

In path tracking, or packet trajectory tracing, information about the path a packet has taken in a network is gathered.

UniRope [248] consists of two different algorithms for packet trajectory tracing that can be selected dynamically to be able to choose the trade-off between accuracy and efficiency. These two algorithms are *compact hash matching* and *consecutive bits filling*. With compact hash matching, the forwarding switch calculates a hash value and stores it in the packet. With consecutive bits filling, the packet trajectory is recorded in the packet hop by hop and reconstructed at the controller.

Knossen et al. [249] present two different approaches for path tracking in P4. In *hop recording*, all forwarding P4 nodes record their ID in the header of the target packet. The last node can then reconstruct the path. In *forwarding state logging*, the first P4 node records the current version of the global forwarding state of the network and its node identifier in a header of the target packet. If the version of the global forwarding state does not change while the packet flows through the network, the last P4 node in the network can reconstruct the path using the information in the header.

Basuki et al. [250] propose a privacy-aware path-tracking mechanism. Their goal is that the trajectory information in the packets cannot be used to draw

conclusions about the network topology or routing information. They achieve this by recording the information in an in-packet bloom filter.

9.7. Other Fields of Application

BurstRadar [251] is a system for microburst detection for data center networks that runs directly on P4 switches. If queue-induced delay is above a certain threshold, BurstRadar reports a microburst and creates a snapshot of the telemetry information of involved packets. This telemetry information is then forwarded to a monitoring server. As it is not possible to gather telemetry information of packets that are already part of the egress queue, the telemetry information of all packets and their corresponding egress port are temporarily stored in a ring buffer that is implemented using P4 registers.

Dapper [253] is a P4 tool to evaluate TCP. It implements TCP in P4 and analyzes header fields, packets sizes, and timestamps of data and ACK packets to detect congestion. Then, flow-dependent information are stored in registers.

He et al. [254] propose an adaptive expiration timeout mechanism for flow entries in P4 switches. The switches implement a mechanism to detect the last packet of a TCP flow. In case of a match, it notifies the controller to delete the corresponding flow entries.

Riesenberg et al. [255] implement alternate marking performance measurement (AM-PM) for P4. AM-PM measures delay and packet loss in-band in a network using only one or two bit overhead per packet. These bits are used for coordination and signalling between measurement points (MPs).

Wang et al. [257] describe how TCP-friendly meters can be designed and implemented for P4-based switches. According to their findings, meters in commercial switches interact with TCP streams in such a way that these streams can only reach about 10% of the target rate. The experimental evaluation of their TCP-friendly meters shows achieved rates of up to 85% of the target rate.

P4STA [258] is an open-source framework that combines software-based traffic load generation with accurate hardware packet timestamps. Thereby, P4STA aggregates multiple traffic flows to generate high traffic load and leverage programmable platforms.

Hark et al. [260] use P4 to filter data plane measurements. To save resources, only relevant measurements are sent to the controller. The authors implement a prototype and demonstrate the system by filtering measurements for a bandwidth forecast application.

P4Entropy [261] presents an algorithm to estimate the entropy of network traffic within the P4 data plane. To that end, they also developed two new algorithms, P4Log and P4Exp, to estimate logarithms and exponential functions within the data plane as well.

Taffet et al. [263] describe a P4-based implementation of an in-band monitoring system that collects information about the path of a packet and whether it encountered congestion. For this purpose, the authors repurpose previously unused fields of the IP header.

NetView [264] is a network telemetry framework that uses proactive probe packets to monitor devices. Telemetry targets, frequency, and characteristics

can be configured on demand by administrators. The probe packets traverse arbitrary paths by using source routing.

FastFE [265] is a system for offloading feature extraction, i.e., deriving certain information from network traffic, for machine learning (ML)-based traffic analysis applications. Policies for feature extraction are defined as sequential programs. A policy enforcement engine translates these policies into primitives for either a programmable switch or a program running on a commodity server.

Unroller [266] detects routing loops in the data plane in real-time. It achieves this by encoding a subset of the path that a packet takes into the packet.

Hang et al. [267] use a time-based sliding window approach to measure packet rates. The goal is to record statistics entirely inside the data plane without having to use the CPU of a switch. Their approach is able to measure traffic size without sampling.

FlowSpy [268] is a network monitoring framework that uses load balancing. Different monitoring tasks are distributed among all available switches by an ILP solver. This reduces the workload on single switches in contrast to monitoring frameworks that perform all monitoring tasks on ingress or egress switches only.

9.8. Summary and Analysis

This research domain greatly benefits from all five core features described in Section 8.1. *Definition and usage of custom packet headers* enables new monitoring schemes where relevant information can be added to packets while it travels through a P4-enabled network. One example is In-band Network Telemetry (INT) (Section 9.4) that has been specified specifically for P4. Another example are path tracking mechanisms (Section 9.6) where the path of a packet is recorded in a dedicated header of the packet. In the case of INT, this goes hand in hand with *flexible packet header processing* as INT headers may contain instructions that other INT-enabled switches need to execute. *Target-specific packet header processing functions* in the form of stateful packet processing using, e.g., registers, is used by all areas of monitoring as it is necessary to gather data over a certain time frame instead of just looking at a single packet. Because the register space is severely limited on most hardware targets, an efficient usage of the available resources is of great importance. Sketches (Section 9.3) is one approach to solve this. After monitoring data is gathered on the control plane, the result is often *processed on the control plane*. This can range from simple notifications to splitting operations between data plane and control plane where the resources on the data plane are not sufficient. Some DSL-based monitoring approaches (Section 9.5) make use of *flexible development and deployment*. With these approaches, a P4 program is generated automatically on the basis of a monitoring workflow defined by an administrator.

10. Applied Research Domains: Traffic Management and Congestion Control

We describe applied research on data center switching, load balancing, congestion notification, traffic scheduling, traffic aggregation, active queue manage-

ment (AQM), and traffic offloading. Table 7 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

10.1. Data Center Switching

Trellis [277, 278] is an open-source multipurpose L2/L3 spine-leaf switch fabric for data center networks. It is designed to run on white box switches in conjunction with the ONOS controller where its main functionality is implemented. It supports typical data center functionality such as bridging using VLANs, routing (IPv4/IPv6 unicast/multicast routing, MPLS segment routing), and vRouter functionality (BGBv4/v6, static routes, route black-holing). Trellis is part of the CORD platform that leverages SDN, network function virtualization (NFV), and Cloud technologies for building agile data centers for the network edge.

DC.p4 [280] implements typical features of data center switches in P4. The list of features includes support for VLAN, NVGRE, VXLAN, ECMP, IP forwarding, access control lists (ACLs), packet mirroring, MAC learning, and packet-in/-out messages to the control plane.

Fabric.p4 is [282, 278] the underlying reference data plane pipeline implemented in P4. By introducing support for P4 switches, the authors aim at increasing the platform heterogeneity for the CORD fabric. Fabric.p4 is currently based on the V1Model switch architecture, but support for PSA is planned. It is inspired by the OpenFlow data plane abstraction (OF-DPA) and currently supports L2 bridging, IPv4/IPv6 unicast/multicast routing, and MPLS segment routing. Fabric.p4 comes with capability profiles such as *fabric* (basic profile), *spgw* (S/PGW), and INT. For control plane interaction, ONOS is extended by the P4Runtime.

RARE [284] (Router for Academia, Research & Education) is developed in the GÉANT project GN4-3 and implements a P4 data plane for the FreeRouter open-source control plane. Its feature list includes routing, bridging, ACLs, VLAN, VXLAN, MPLS, GRE, MLDP, and BIER among others.

10.2. Load Balancing

SHELL [286] implements stateless application-aware load balancing in P4. A load balancer forwards new connections to a set of randomly chosen application instances by adding a segment routing (SR) header. Each application instance makes a local decision to either decline or accept the connection attempt. After connection initiation, the client includes a previously negotiated identifier in all subsequent packets. In the prototypical implementation, the authors use TCP time stamps for communicating the identifier, alternatives are identifiers of QUIC or TCP sequence numbers.

SilkRoad [287] implements stateful load balancing on P4 switches. SilkRoad implements two tables for stateful processing. One table maps virtual IP addresses of services to server instances, another table records active connections identified by hashes of 5-tuples to forward subsequent flows. It applies a Bloom

Table 7: Overview of applied research on traffic management and congestion control (Section 10).

Research work	Year	Targets	Code
Data Center Switching (Section 10.1)			
Trellis [277, 278]	2019	bmv2	[279]
DC.p4 [280]	2015	bmv2	[281]
Fabric.p4 [282]	2018	bmv2	[283]
RARE [284]	2019	bmv2, Tofino	[285]
Load Balancing (Section 10.2)			
SHELL [286]	2018	NetFPGA-SUME	
SilkRoad [287]	2017	Tofino	
HULA [288]	2016	-	
MP-HULA [289]	2018	-	
Chiang et al. [290]	2019	bmv2	
W-ECMP [291]	2018	bmv2	
DASH [292]	2020	bmv2	
Pizzutti et al. [293, 294]	2018/20	bmv2	
LBAS [295]	2020	Tofino	
DPRO [296]	2020	bmv2	
Kawaguchi et al. [297]	2019	bmv2	
AppSwitch [298]	2017	PISCES	
Beamer [299]	2018	bmv2, NetFPGA-SUME	[300]
Congestion Notification (Section 10.3)			
P4QCN [301]	2019	bmv2	
Jiang et al. [302]	2019	-	
EECN [303]	2020	bmv2	
Chen et al. [304]	2020	bmv2	
Laraba et al. [305]	2020	bmv2	
Traffic Scheduling (Section 10.4)			
Sharma et al. [306]	2018	bmv2	
Cascone et al. [307]	2017	-	
Bhat et al. [308]	2019	bmv2	
Kfoury et al. [309]	2019	bmv2	
Chen et al. [310]	2019	Tofino	
Lee et al. [311]	2019	bmv2	
Traffic Aggregation (Section 10.5)			
Wang et al. [312]	2020	Tofino	
RL-SP-DRR [313]	2019	bmv2	

Research work	Year	Targets	Code
Active Queue Management (AQM) (Section 10.6)			
Turkovic et al. [314]	2018	bmv2, Netronome	
P4-Codel [315]	2018	bmv2	[316]
P4-ABC [317]	2019	bmv2	
P4air [318]	2020	bmv2, Tofino	
Fernandes et al. [319]	2020	bmv2	
Wang et al. [320]	2018	bmv2, Tofino	
SP-PIFO [321]	2020	Tofino	
Kunze et al. [322]	2021	Tofino	[323]
Harkous et al. [324]	2021	bmv2, Netronome	
Traffic Offloading (Section 10.7)			
Andrus et al. [325]	2019	-	
Ibanez et al. [326]	2019	NetFPGA-SUME	
Kfoury et al. [327]	2020	Tofino	
Falcon [328]	2020	Tofino	
Osiński et al. [329]	2020	Tofino	

filter to identify new connection attempts and to record those requests in registers to remember client requests that arrive while the pool of server instances changes. In [330], the accompanying demo is described.

HULA [288] implements a link load-based distance vector routing mechanism. Switches in HULA do not maintain the state for every path but the next hops. They send out probes to gather link utilization information. Probe packets are distributed throughout the network on node-specific multicast trees. The probes have a header that contains a destination field and the currently best path utilization to that destination. When a node receives a probe, it updates the best path utilization if necessary, sends one packet clone upstream back to the origin, and forwards copies along the multicast tree further downstream. This way the origin will receive multiple probe packets with different path utilization to a specific destination. Then, flowlets are forwarded onto the best currently available path to its destination.

MP-HULA [289] extends HULA by using load information for n best next hops and compatibility with multipath TCP (MP-TCP). It tracks subflows of MP-TCP with individual flowlets per sub-flow. MP-HULA aims at distributing those subflows on different paths to aggregate bandwidth. To that end, it is necessary to keep track of the best n next-hops which is done with additional registers and forwarding rules.

Chiang et al. [290] propose a cost-effective congestion-aware load balancing scheme (CCLB). In contrast to HULA, CCLB replaces only the leaf switches with programmable switches, and thus is more cost-effective. They leverage Explicit Congestion Notification (ECN) information in probe packets to recognize

congestion in the network and to adapt the load balancing. CCLB further uses flowlet forwarding and is implemented for the bmv2.

W-ECMP [291] is an ECMP-based load balancing mechanism for data centers implemented for P4 switches. Weighted probabilities based on path utilization, are used to randomly choose the best path to avoid congestion. A local agent on each switch computes link utilization for the ports. Regular traffic carries an additional custom packet header that keeps track of the current maximum link utilization on a path. Based on the maximum link utilization, the switches update port weights if necessary.

DASH [292] is an adaptive weighted traffic splitting mechanism that works entirely in the data plane. In contrast to popular weighted traffic splitting strategies such as WCMP, DASH does not require multiple hash table entries. DASH splits traffic based on link weights by portioning the hash space into unique regions.

Pizzutti et al. [293, 294] implement congestion-aware load balancing for flowlets on P4 switches. Flowlets are bursts of packets that are separated by a time gap, e.g., as caused by factors such as TCP dynamics, buffer availability, or link congestion. For distributing subflows on different paths, the congestion state of the last route is stored in a register.

LBAS [295] implements a load balancer to minimize the processing latency at both load balancers and application servers. LBAS does not only reduce the processing latency at load balancers but also takes the application servers' state into account. It is implemented for the Tofino and its average response time is evaluated.

DPRO [296] combines INT with traffic engineering (TE) and reinforcement learning (RL). Network statistics, such as link utilization and switch load, are gathered using an adapted INT approach. An RL-agent inside the controller adapts the link weights based on the minimization of a max-link-utilization objective.

Kawaguchi et al. [297] implement Unsplittable flow Edge Load factor Balancing (UELB). A controller application monitors the link utilization and computes new optimal paths upon congestion. The path computation is based on the UELB problem. The forwarding is implemented in P4 for the bmv2.

AppSwitch [298] implements a load balancer for key-value storage systems. However, the focus lies on a local agent and the control plane communication with the storage server.

Beamer [299] operates in data centers and prevents interruption of connections when they are load-balanced to a different server. To that end, the Beamer controller instructs the new target server to forward packets of the load-balanced connection to the old target server until the migration phase is over.

10.3. Congestion Notification

P4QCN [301] proposes a congestion feedback mechanism where network nodes check the egress ports for congestion before forwarding packets. If a node detects congestion, it calculates a feedback value that is propagated upstream. The mechanism clones the packet that caused the congestion, updates

the feedback value in the header, changes the origin of the flow, and forwards it as a feedback packet to the sender. The sender adjusts its sending rate to reduce congestion downstream. The authors describe an implementation where bmv2 is extended by P4 externs for floating-point calculations.

Jiang et al. [302] introduce a novel adjusting advertised windows (AWW) mechanism for TCP. The authors argue that the current calculation of the advertised window in the TCP header is inaccurate because the source node does not know the actual capacity of the network. AWW dynamically updates the advertised window of ACK packets to feedback the network capacity indirectly to the source nodes. Each P4 switch calculates the new AWW value and writes it into the packet header.

EECN [303] presents an enhanced ECN mechanism which piggybacks congestion information if the switch notices congestion. To that end, the ECN-Echo bit is set for traversing ACKs as soon as congestion occurs for a given flow. This enables fast congestion notification without the need for additional control traffic.

Chen et al. [304] present QoSTCP, a TCP version with adapted congestion window growth that enables rate limiting. QoSTCP is based on a marking approach similar to ECN. When a flow exceeds a certain rate, the packet gets marked with a so-called Rate-Limiting Notification (RLN) and the congestion window growth is adapted proportional to the RLN-marked packet rate. Metering and marking is done using P4.

Laraba et al. [305] detect ECN misbehavior with the help of P4 switches. They model ECN as extended finite state machine (EFSM) and store states and variables in registers. If end hosts do not conform to the specified ECN state machine, packets are either dropped or, if possible, the misbehavior is corrected.

10.4. Traffic Scheduling

Sharma et al. [306] introduce a mechanism for per flow fairness scheduling in P4. The concept is based on round-robin scheduling where each flow may send a certain number of bytes in each round. The switch assigns a round number for each arriving packet that depends on the number of sent bytes of flow in the past.

Cascone et al. [307] introduce bandwidth sharing based on sending rates between TCP senders. P4 switches use statistical byte counters to store the sending rate of each user. Depending on the recorded sending rate of the user, arriving packets are pushed into different priority queues.

Bhat et al. [308] leverage P4 switches to translate application layer header information into link-layer headers for better QoS routing. They use Q-in-Q tunneling at the edge to forward packets to the core network and present a bmv2 implementation for HTTP/2 applications, as HTTP/2 explicitly defines a Stream ID that can directly be translated in Q-in-Q tags.

Kfoury et al. [309] present a method to support dynamic TCP pacing with the aid of network state information. A P4 switch monitors the number of active TCP flows, i.e., they monitor the SYN, SYN-ACK, and ACK flags and

notify senders about the current network state if a new flow starts or another terminates. To that end, they introduce a new header and show by simulations that the overall throughput increases.

Chen et al. [310] present a design for bandwidth management for QoS with SDN and P4-programmable switches. Their design classifies packets based on a two-rate three-color marker and assigns corresponding priorities to guarantee certain per flow bandwidth. To that end, they leverage the priority queuing capabilities of P4-switches based on the assigned color. Guaranteed traffic goes to a high-priority queue, best-effort traffic goes to a low-priority queue, and traffic that exceeds its bandwidth is simply dropped.

Lee et al. [311] implement a multi-color marker for bandwidth guarantees in virtual networks. Their objective is to isolate bandwidth consumption of virtual networks and provide QoS for its serving flows.

10.5. Traffic Aggregation

Wang et al. [312] introduce aggregation and dis-aggregation capabilities for P4 switches. To reduce the header overhead in the network, multiple small packets are thereby aggregated to a single packet. They leverage multiple register arrays to store incoming small packets in 32 bit chunks. If enough small packets are stored, a larger packet gets assembled with the aid of multiple recirculations; each recirculation step appends a small packet to the aggregated large packet.

RL-SP-DRR [313] is a combination of strict priority scheduling with rate limitation (RL-SP) and deficit round-robin (DRR). RL-SP ensures prioritization of high-priority traffic while DRR enables fair scheduling among different priority classes. They extend bmv2 to support RL-SP-DRR and evaluate it against strict priority queuing and no active queuing mechanism.

10.6. Active Queue Management (AQM)

Turkovic et al. [314] develop an active queue management (AQM) mechanism for programmable data planes. The switches are programmed to collect metadata associated with packet processing, e.g., queue size and load, that are used to prevent, detect, and dissolve congestion by forwarding affected flows on an alternate path. Two possible mechanisms for rerouting in P4 are described. In the first mechanism, primary and backup entries are installed in the forwarding tables and according to the gathered metadata, the suitable action is selected. The second mechanism leverages a local controller on each switch that monitors flows and installs updated forwarding rules when congestion is noticed.

P4-CoDel [315] implements the CoDel AQM mechanism specified in RFC 8289 [331]. CoDel leverages a target and an interval parameter. As long as the queuing delay is shorter than the target parameter, no packets are dropped. If the queuing delay exceeds the target by a value that is at least as large as the interval, a packet is dropped, and the interval parameter is decreased. This procedure is repeated until the queuing delay is under the target threshold again. The interval is then reset to the initial value. To avoid P4 externs, the authors use approximated calculations for floating-point operations.

P4-ABC [317] implements activity-based congestion management (ABC) for P4. ABC is a domain concept where edge nodes measure the activity, i.e., the sending rate, of each user and annotate the value in the packet header. Core nodes measure the average activity of all packets. Depending on the current queue status, the average activity, and activity value in the packet header, a drop decision is made for each packet to prevent congestion. The $P4_{16}$ implementation for the bmv2 requires externs for floating-point calculations.

P4air [318] attempts to provide more fairness for TCP flows with different congestion control algorithms. To that end, P4air groups flows into different categories based on their congestion control algorithm, e.g., loss-, delay- and loss-delay-based. Afterwards, the most aggressive flows are punished based on the previous categorization with packet drops, delay increase, or adjusted receive windows. P4air leverages switch metrics and flow reactions, such as queuing delay and sending rate, to determine the congestion control algorithm used by the flows.

Fernandes et al. [319] propose a bandwidth throttling solution in P4. Incoming packets are dropped with a certain probability depending on the incoming rate of the flow and the defined maximum bandwidth. Rates are measured using time windows and byte counters. Fernandes et al. extend the bmv2 for this purpose.

Wang et al. [320] present an AQM mechanism for video streaming. Data packets are classified as base packets (basic image information) or enhancement packets (additional information to improve the image quality). When the queue size exceeds a certain threshold, enhancement packets are preferably dropped.

SP-PIFO [321] features an approximation of Push-In First-Out (PIFO) queues which enables programmable packet scheduling at line rate. SP-PIFO dynamically adapts the mapping between packet ranks and available strict-priority queues.

Kunze et al. [322] analyze the design of three popular AQM algorithms (RED, CoDel, PIE). They implement PIE in three different variants for Tofino-based P4 hardware targets and show that implementation trade-offs have significant performance impacts.

Harkous et al. [324] use virtual queues implemented in P4 for traffic management. A traffic classifier in the form of MATs assigns a data plane slice identifier to traffic flows. P4 registers are used to implement virtual queues for each data plane slice for traffic management.

10.7. Traffic Offloading

Andrus et al. [325] propose to offload video stream processing of surveillance cameras to P4 switches. The authors propose to offload stream processing for storage to P4 switches. In case the analytics software detected an event, it enables a multistage pipeline on the P4 switch. In the first step, video stream data is replicated. One stream is further sent to the analytics software, the other stream is dedicated to the video storage. The P4 switch filters out control packets and rewrites the destination IP address of all video packets to the video storage.

Ibanez et al. [326] try to tackle the problem of P4's packet-by-packet programming model. Many tasks, such as periodic updates, require either hardware-specific capabilities or control-plane interaction. Processing capabilities are limited to enqueue events, i.e., data plane actions are only triggered if packets arrive. To eliminate this problem, the authors propose a new mechanism for event processing using the P4 language.

Kfoury et al. [327] propose to offload media traffic to P4 switches which act as relay servers. A SIP server receives the connection request, replaces IP and port information with the relay server IP and port, and forwards the request to the receiver. Afterwards, the media traffic is routed through the relay server.

Falcon [328] offloads task scheduling to programmable switches. Job requests are sent to the switch and the switch assigns a task in first-come-first-serve order to the next executor in a pool of computation nodes. Falcon reduces the scheduling overhead by a factor of 26 and increase scheduling throughput by a factor of 25 compared to state-of-the-art schedulers.

Osinski et al. [329] present vBNG, a virtual Broadband Network Gateway (BNG). Some components, such as PPPoE session handling, are offloaded to programmable switches.

10.8. Summary and Analysis

The research domain of traffic management and congestion control benefits from three core properties of P4: *custom packet headers*, *flexible header processing* and *target-specific packet header processing functions*. Data center switching mainly relies on packet header parsing of well-known protocols, such as IPv4/v6 or MPLS. More advanced protocol solutions, such as VXLAN and BIER, can be implemented by leveraging the *flexible packet header processing* property of P4. The presented efforts on load balancing (Section 10.2) also use this property of P4 to implement novel approaches. *Target-specific packet header processing functions* such as externs are widely used in Section 10.3. Most works leverage externs such as metering and marking which may not be supported on all hardware targets. A similar phenomenon appears in Section 10.4. Here, many papers are based on priority queues. The approaches on AQM in Section 10.6 encounter similar limitations. Floating-point operations are not part of the P4 core. Some targets may provide an extern for this functionality. Multiple works avoid this problem by either using approximations or by relying on self-defined externs in software.

11. Applied Research Domains: Routing and Forwarding

We describe applied research on source routing, multicast, publish-subscribe-systems, named data networking, data plane resilience, and other fields of application. Table 8 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

Table 8: Overview of applied research on routing and forwarding (Section 11).

Research work	Year	Targets	Code
Source Routing (11.1)			
Lewis et al. [332]	2018	bmv2	[333]
Luo et al. [334]	2019	bmv2	[335]
Kushwaha et al. [336]	2020	Xilinx Virtex-7	
Abdelsalam et al. [337]	2020	bmv2	
Multicast (11.2)			
Braun et al. [338]	2017	bmv2	[339]
Merling et al. [340, 341]	2020/21	bmv2, Tofino	[342, 343]
Elmo [344]	2019	-	[345]
PAM [346]	2020	bmv2	
Publish/Subscribe Systems (11.3)			
Wernecke et al. [347, 348, 349, 350]	2018/19	bmv2	
Jepsen et al. [351]	2018	Tofino	
Kundel et al. [352]	2020	bmv2	[353]
FastReact-PS [354]	2020	-	
Named Data Networks (11.4)			
NDN.p4 [355, 356]	2016/18	bmv2	[357, 358]
ENDN [359]	2020	bmv2	
Data Plane Resilience (11.5)			
Sedar et al. [360]	2018	bmv2	[361]
Giesen et al. [362]	2018	Tofino, Xilinx SDNet	
SQR [363]	2019	bmv2, Tofino	[364]
P4-Protect [365]	2020	bmv2, Tofino	[366, 367]
Hirata et al. [368]	2019	-	
Lindner et al. [369]	2020	bmv2, Tofino	[370, 371]
D2R [372]	2019	bmv2	
PURR [373]	2019	bmv2, Tofino	
Blink [374]	2019	bmv2, Tofino	[375]

Research work	Year	Targets	Code
Other Fields of Applications (11.6)			
Contra [376]	2019	-	
Michel et al. [377]	2016	bmv2	
Baktir et al. [378]	2018	bmv2	
Froes et al. [379]	2020	bmv2	
QROUTE [380]	2020	bmv2	
Gimenez et al. [381]	2020	bmv2	
Feng et al. [382]	2019	bmv2	
PFCA [383]	2020	bmv2	
McAuley et al. [384]	2019	bmv2	
R2P2 [385]	2019	Tofino	[386]

11.1. Source Routing

With source routing, the source node defines the processing of the packet throughout the network. To that end, a header stack is often added to the packet to specify the operations the other network devices should execute.

Lewis et al. [332] implement a simple source routing mechanism with P4 for the bmv2. The authors introduce a header stack to specify the processing of the packet towards its destination. That header stack is constructed and pushed onto the packet by the source node. Network devices match the header segments to determine how the packet should be processed.

Luo et al. [334] implement segment routing with P4. They introduce a header which contains segments that identify certain operations, e.g., forwarding the packet towards a specific destination or over a specific link, updating header fields, etc. Network nodes process packets according to the topmost segment in the segment routing header and remove it after successful execution.

Kushwaha et al. [336] implement bitstream, a minimalistic programmable data plane for carrier-class networks, in P4 for FPGAs. The focus of bitstream is to provide a programmable data plane while ensuring several carrier-grade properties, like deterministic latencies, short restoration time, and per-service measurements. To that end, the authors implement a source routing approach in P4 which leaves the configuration of the header stack to the control plane.

The authors of [337] show a demo of segment routing over IPv6 data plane (SRv6) implementation in P4. It leverages the novel uSID instruction set for SRv6 to improve scalability and MTU efficiency.

11.2. Multicast

Multicast efficiently distributes one-to-many traffic from the source to all subscribers. Instead of sending individual packets to each destination, multicast packets are distributed in tree-like structures throughout the network.

Bit Index Explicit Replication (BIER) [387] is an efficient transport mechanism for IP multicast traffic. In contrast to traditional IP multicast, it prevents

subscriber-dependent forwarding entries in the core network by leveraging a BIER header that contains all destinations of the BIER packet. To that end, the BIER header contains a bit string where each bit corresponds to a specific destination. If a destination should receive a copy of the BIER packet, its corresponding bit is activated in the bit string in BIER header of the packet. Braun et al. [338] present a demo implementation of BIER-based multicast in P4. Merling et al. [340] implement BIER-based multicast with fast reroute capabilities in P4 for the bmv2 and for the Tofino [341].

Elmo [344] is a system for scalable multicast in multi-tenant datacenters. Traditional IP multicast maintains subscriber dependent state in core devices to forward multicast traffic. This limits scalability, since the state in the core network has to be updated every time subscribers change. Elmo increases scalability of IP multicast by moving a certain subscriber-dependent state from the core devices to the packet header.

Priority-based adaptive multicast (PAM) [346] is a control protocol for data center multicast which is implemented by the authors in P4. Network administrators define different policies regarding priority, latency, completion time, etc., which are installed on the core switches. The network devices than monitor link loads and adjust their forwarding to fulfill the policies.

11.3. Publish/Subscribe Systems

Publish/subscribe systems are used for data distribution. Subscribers are able to subscribe to announced topics. Based on the subscriptions, the data packets are distributed from the source to all subscribers.

Wernecke et al. [347, 348, 349, 350] implement a content-based publish/subscribe mechanism with P4. The distribution tree to all subscribers is encoded directly in the header of the data packets. To that end, the authors introduce a header stack which is pushed onto the packet by the source. Each element in the stack consists of an ID and a value. When a node receives a packet, it checks whether the header stack contains an element with its own ID. If so, the value determines to which neighbors the packet has to be forwarded.

Jepsen et al. [351] introduce a description language to implement publish/subscriber systems. The data plane description is translated into a static pipeline and dynamic filters. The static pipeline is a P4 program that describes a packet processing pipeline for P4 switches, the dynamic filters are the forwarding rules of the match-action tables that may change during operation, e.g., when subscriptions change.

Kundel et al. [352] propose two approaches for attribute/value encoding in packet headers for P4-based publish/subscribe systems. This reduces the header overhead and facilitates adding new attributes which can be used for subscription by hosts.

FastReact-PS [354] is a P4-based framework for event-based publish/subscribe in industrial IoT networks. It supports stateful and stateless processing of complex events entirely in the data plane. Thereby, the forwarding logic can be dynamically adjusted by the control plane without the need for recompilation.

11.4. Named Data Networking

Named data networking (NDN) is a content-centric paradigm where information is requested with resource identifiers instead of destinations, e.g., IP addresses. Network devices cache recently requested resources. If a requested resource is not available, network devices forward the request to other nodes.

NDN.p4 [355] implements NDN without caching for P4. However, the implementation cannot cache requests because of P4-related limitations with stateful storage. Miguel et al. [356] leverage the new functionalities of $P4_{16}$ to extend NDN.p4 by a caching mechanism for requests and optimize its operation. The caching mechanism is implemented with P4 externs.

Enhanced NDN (ENDN) [359] is an advanced NDN architecture. It offers a larger catalog of content delivery features like adaptive forwarding, customized monitoring, in-network caching control, and publish/subscribe forwarding.

11.5. Data Plane Resilience

Sedar et al. [360] implement a fast failover mechanism without control plane interaction for P4 switches. The mechanism uses P4 registers or metadata fields for bit strings that indicate if a particular port is considered up or down. In a match-action table, the port bit string provides an additional match field to determine whether a particular port is up or down. Depending on the port status, default or backup actions are executed. The authors rely on a local P4 agent to populate the port bit strings.

Giesen et al. [362] introduce a forward error correction (FEC) mechanism for P4. Commonly, unreliable but not completely broken links are avoided. As this happens at the cost of throughput, the proposed FEC mechanism facilitates the usage of unreliable links. The concept features a link monitoring agent that polls ports to detect unreliable connections. When a packet should be forwarded over such a port, the P4 switch calculates a resilient encoding for the packet which is then decoded by the receiving P4 switch.

Shared Queue Ring (SQR) [363] introduces an in-network packet loss recovery mechanism for link failures. SQR caches recent traffic inside a queue with slow processing speed. If a link failure is detected, the cached packets can be sent over an alternative path. While P4 does not offer the possibility to store packets for a certain amount of time, the authors leverage the cloning operation of P4 to keep packets inside the buffer. If a cached packet has not yet met its delay, it gets cloned to another egress port which takes some time. This procedure is repeated until the packet has been stored for a given time span.

P4-Protect [365] implements 1+1 protection for IP networks. Incoming packets are equipped with a sequence number, duplicated, and sent over two disjoint paths. At an egress point, the first version of each packet is accepted and forwarded. As a result, a failure of a single path can be compensated without additional signaling or reconfiguration. P4-Protect is implemented for the bmv2 and the Tofino. Evaluations show that line-rate processing with 100 Gbit/s can be achieved with P4-Protect at the Tofino.

Hirata et al. [368] implement a data plane resilience scheme based on multiple routing configurations. Multiple routing configurations with disjoint paths

are deployed, and a header field identifies the routing configuration according to which packets are forwarded. In the event of a failure, a routing configuration is chosen that avoids the failure.

Lindner et al. [369] present a novel prototype for in-network source protection in P4. A P4-capable switch receives sensor data from a primary and secondary sensor, but forwards only the data from the primary sensor if available. It detects the failure of the primary sensor and then transparently forwards data from a secondary sensor to the application. Two different mechanisms are presented. The *counter-based* approach stores the number of packets received from the secondary sensor since the last packet from the primary sensor has been received. The *timer-based* approach stores the time of the last arrival of a packet from the primary sensor and considers the time since then. If certain thresholds are exceeded, the P4-switch forwards the data from the secondary sensor.

D2R [372] is a data-plane-only resilience mechanism. Upon a link failure, the data plane calculates a new path to the destination using algorithms like breadth-first search and iterative deepening depth-first search. As one pipeline iteration has not enough processing stages to compute the path, recirculation is leveraged. In addition, *Failure Carrying Packets (FCP)* is used to propagate the link failure inside the network. While the authors claim that their architecture works with hardware switches, e.g., the Tofino, they only present and evaluate a bmv2 implementation.

Chiesa et al. [373] propose a primitive for reconfigurable fast ReRoute (PURR) which is a FRR primitive for programmable data planes, in particular for P4. For each destination, suitable egress ports are stored in bit strings. During packet processing, the first working suitable egress port is determined by a set of forwarding rules. Encoding based on *Shortest Common Supersequence* guarantees that only few additional forwarding rules are required.

Blink [374] detects failures without controller interaction by analyzing TCP signals. The core concept is that the behavior of a TCP flow is predictable when it is disrupted, i.e., the same packet is retransmitted multiple times. When this information is aggregated over multiple flows, it creates a characteristic failure signal that is leveraged by data plane switches to trigger packet rerouting to another neighbor.

11.6. Other Fields of Applications

Contra [376] introduces performance-aware routing with P4. Network paths are ranked according to policies that are defined by administrators. Contra applies those policies and topology information to generate P4 programs that define the behavior of forwarding devices. During runtime, probe packets are used to determine the current network state and update forwarding entries for best compliance with the defined policies.

Michel et al. [377] introduce identifier-based routing with P4. The authors argue that IP addresses are not fine-granular enough to enable adequate forwarding, e.g., in terms of security policies. The authors introduce a new header

that contains an identifier token. Before sending packets, applications transmit information on the process and user to a controller that returns an identifier that is inserted into the packet header. P4 switches are programmed to forward packets based on that identifier.

Baktir et al. [378] propose a service-centric forwarding mechanism for P4. Instead of addressing locations, e.g., by IP addresses, the authors propose to use location-independent service identifiers. Network hosts write the identifier of the desired service into the appropriate header field, the switches then make forwarding decisions based on the identifier in the packet header. With this approach, the location of the service becomes less important since the controller simply updates the forwarding rules when a service is migrated or load balancing is desired.

Froes et al. [379] classify different traffic classes which are identified by a label. Packet forwarding is based on that controller-generated label instead of IP addresses. The traffic classes have different QoS properties, i.e., prioritization of specific classes is possible. To that end, switches leverage multiple queues to process traffic of different traffic classes.

QROUTE [380] is a quality of service (QoS) oriented forwarding scheme in P4. Network devices monitor their links and annotate values, e.g., jitter or delay, in the packet header so that downstream nodes can update their statistics. Furthermore, packet headers contain constraints like maximum jitter or delay. According to those values, forwarding decisions are made by the network devices.

Gimenez et al. [381] implement the recursive internet-work architecture (RINA) in P4 for the bmv2. RINA is a networking architecture which sees computer networking as a type of inter-process communication where layering should be based on scope/scale instead of function. In general, efficient implementations require hardware support. However, up to date only software-based implementations are available. The authors hope that with the advance of programmable hardware in the form of P4, hardware-based RINA will soon be possible.

Feng et al. [382] implement information-centric network (ICN) based forwarding for HTTP. To that end, they propose mechanisms to convert packets from ICN to HTTP packets and vice-versa.

PFCA [383] implements a forwarding information base (FIB) caching architecture in the data plane. To that end, the P4 program contains multiple MATs that are mapped to different memory, i.e., TCAM, SRAM, dynamic random access memory (DRAM), with different properties regarding lookup speed. Counters keep track of cache hits to move (un)popular rules to other tables.

McAuley et al. [384] present a hybrid error control booster (HEC) that can be deployed in wireless, mobile, or hostile networks that are prone to link or transport layer failures. HECs increase the reliability by applying a modified Reed-Solomon code that adds parity packets or additional packet block acknowledgments. P4 targets include an error control processor that implements this functionality. It is integrated into the P4 program as P4 extern so that the data plane can exchange HEC packets with it. A remote control plane includes the booster manager that controls HEC operations and parameters on the P4

targets via a data plane API.

R2P2 [385] is a transport protocol based on UDP for latency-critical RPCs optimized for datacenters or other distributed infrastructure. A router module implemented in P4 or DPDK is used to relay requests to suitable servers and perform load balancing. It may also perform queuing if no suitable server is available. The goal of R2P2 is to overcome problems that typically come with TCP-based RPC systems, e.g., problems with load distribution and head-of-line-blocking.

11.7. Summary and Analysis

The research domain of routing and forwarding greatly benefits from P4's core features. First, the *definition and usage of custom packet headers* enables administrators to tailor the packet header to the specific use case. Two examples are source routing (Section 11.1) and multicast (Section 11.2). Both areas leverage custom headers to define lightweight mechanisms based on additional information in the packet header which are not part of any standard protocol. Although most of the projects were developed only for the bmv2, they should be easily portable to hardware platforms as more complex, target specific operations are not required. Second, users are able to define *flexible packet header processing* depending on the information in the packet header, e.g., publish/-subscribe systems (Section 11.3), named data networks (Section 11.4), and data plane resilience (Section 11.5). Parametrized custom actions and (conditional) application of multiple MATs allow for adaptable packet processing for many specific use cases. Similar to the previous P4 core feature, most projects were developed for the bmv2 but they should be easy to transfer if no target-specific actions are used. Third, we found that many papers in the area of data plane resilience (Section 11.5) leverage *target-specific packet header processing functions.* Often registers are used to store information whether egress ports are up or down to execute backup actions if necessary. Most projects were implemented for the hardware platform Tofino. As a result, the implementations are highly target-specific and transferring them to other hardware platforms highly depends on the capabilities of the target platform and the used externs.

12. Applied Research Domains: Advanced Networking

We describe applied research on cellular networks (4G/5G), Internet of things (IoT), industrial networking, Time-Sensitive Networking (TSN), network function virtualization (NFV), and service function chains (SFCs). Table 9 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

12.1. Cellular Networks (4G/5G)

P4EC [388] builds a local exit for LTE deployments with cloud-based EPC services. A programmable switch distinguishes traffic and reroutes traffic for edge computing. Non-critical traffic is forwarded to the cloud-based EPC.

Table 9: Overview of applied research on advanced networking (Section 12).

Research work	Year	Targets	Code
Cellular Networks (4G/5G) (12.1)			
P4EC [388]	2020	Tofino	
Trellis [282]	-	-	[389]
SMARTHO [390]	2018	bmv2	
Aghdai et al. [391, 392]	2018/19	Netronome	
GRED [393]	2019	bmv2	
HDS [394]	2020	-	
Shen et al. [395]	2019	Xilinx SDNet	
Lee et al. [396]	2019	Tofino	
Ricart-Sanchez et al. [397]	2019	NetFPGA-SUME	
Singh et al. [398]	2019	Tofino	
TurboEPC [399]	2020	Netronome	
Vörös et al. [400]	20200	Tofino	
Lin et al. [401]	2019	Tofino	
Internet of Things (12.2)			
BLESS [402]	2017	PISCES	
Muppet [403]	2018	PISCES	
Wang et al. [404]	2019	Tofino	
Madureira et al. [405]	2020	bmv2	
Engelhard et al. [406]	2019	bmv2	
Industrial Networking (12.3)			
FastReact [407]	2018	bmv2	
Cesen et al. [408]	2020	bmv2	
Kunze et al. [409]	2020	Tofino, Netronome	
Time-Sensitive Networking (TSN) (12.4)			
Rüth et al. [410]	2018	Netronome	
Kannan et al. [411]	2019	Tofino	
Kundel et al. [412]	2019	Tofino	

Research work	Year	Targets	Code
Network Function Virtualization (NFV) (12.5)			
Kathará [413]	2018	-	
P4NFV [414]	2018	bmv2	
Osiński et al. [415]	2019	-	
Moro et al. [416]	2020	-	
DPPx [417]	2020	bmv2	
Mohammadkhan et al. [418]	2019	Netronome	
FOP4 [419, 420]	2019	bmv2, eBPF	
PlaFFE [421]	2020	Netronome	
Service Function Chains (SFCs) (12.6)			
P4SC [422, 423]	2019	bmv2, Tofino	[424]
Re-SFC [425]	2019	bmv2	
FlexMesh [426]	2020	bmv2	
P4-SFC [427]	2019	bmv2, Tofino	[428]

The Trellis switch fabric (introduced in Section 10.1) features the spgw.p4 profile [282, 278], an implementation of a Serving and PDN Gateway (SPGW) for 5G networking. ONOS runs an SPGW-u application that implements the 3GPP control and user plane separation (CUPS) protocol to create, modify, and delete GPRS tunneling protocol (GTP) sessions. It provides support for GTP en- and decapsulation, filtering, and charging.

SMARTHO [390] proposes a handover framework for 5G. Distributed units (DUs) include real-time functions for multiple 5G radio stations. Several DUs are controlled by a central unit (CU) that includes non-real-time control functions. P4 switches are part of the CU and all DU nodes. SMARTHO introduces a P4-based mechanism for preparing handover sequences for user devices that take a fixed path among 5G radio stations controlled by DUs. This decreases the overall handover time, e.g., for users traveling in a train.

Aghdai et al. [391] propose a P4-based transparent edge gateway (EGW) for mobile edge computing (MEC) in LTE or 5G networks. Delay-sensitive and bandwidth-intense applications need to be moved from data centers in the core network to the edge of the radio access network (RAN). 5G networks rely on GTP-U for encapsulating IP packets from the mobile user to the core network. IP routers in between forward packets based on the outer IP address of GTP-U frames. The authors deploy EGWs as P4 switches at the edge of the IP transport network where service operators can deploy scalable network functions or services. Each MEC service gets a virtual IP address, the P4-based EGWs parse the inner IP destination address of GTP-U. If it sees traffic targeting a virtual IP address of a MEC service, it forwards it to the IP address of one of the serving instances of the MEC application. In their follow-up work [392], the

authors extend EGWs by a handover mechanism for migrating network state.

GRED [393] is an efficient data placement and retrieval service for edge computing. It tries to improve routing path lengths and forwarding table sizes. They follow a greedy forwarding approach based on DT graphs, where the forwarding table size is independent of the network size and the number of flows in the network. GRED is implemented in P4, but the authors do not specify on which target.

HDS [394] is a low-latency, hybrid, data sharing framework for hierarchical mobile edge computing. The data location service is divided into two parts: intra-region and inter-region. The authors present a data sharing protocol called Cuckoo Summary for fast data localization for the intra-region part. Further, they developed a geographic routing scheme to achieve efficient data location with only one overlay hop in the inter-region part.

Shen et al. [395] present an FGPA-based GTP engine for mobile edge computing in 5G networks. Communication between the 5G back-haul and the conventional Ethernet requires de- and encapsulation of traffic with GTP. As most network entities do not have the capability to process GTP, the authors leverage P4-programmable hardware for this purpose.

Lee et al. [396] evaluate the performance of GTP-U and SRv6 stateless translation as GPT-U cannot be replaced by SRv6 without a transition period. To that end, they implement GTP and SRv6 on P4-programmable hardware. They found that there are no performance drops if stateless translation is used and that SRv6 stateless translation is acceptable for the 5G user plane.

Ricart-Sanchez et al. [397] propose an extension for the P4-NetFPGA framework for network slicing between different 5G users. The authors extend the capabilities of the P4 pipeline and implement their mechanism on the NetFPGA-SUME. However, the authors do not provide any details about their implementation.

Singh et al. [398] present an implementation for the Evolved Packet Gateway (EPG) in the Mobile Packet Core of 5G. They show that they can offload the functionality to programmable switching ASICs and achieve line rate with low latency and jitter while scaling up to 1.7 million active users.

TurboEPC [399] presents a redesign of the mobile packet core where parts of the control plane state is offloaded to programmable switches. State is stored in MATs. The switches then process a subset of signaling messages within the data plane itself, which leads to higher throughput and reduced latency.

Vörös et al. [400] propose a hybrid approach for the next generation NodeB (gNB) where the majority of packet processing is done by a high-speed P4-programmable switch. Additional functions, such as ARQ or ciphering, are offloaded to external services such as DPDK implementations.

Lin et al. [401] enhance the Content Permutation Algorithm (eCPA) for secret permutation in 5G. Packet payloads are split into code words and shuffled according to a secret cipher. They implement eCPA for switches of the Inventec D5264 series.

12.2. Internet of Things (IoT)

BLESS [402] implements a Bluetooth low energy (BLE) service switch based on P4 that acts as a proxy enabling flexible, policy-based switching and in-network operations of IoT devices. BLE devices are strictly bound to a central device such as a smartphone or tablet. IoT usage requires cloud-based solutions where central devices connect to an IoT infrastructure. The authors propose a BLE service switch (BLESS) that is transparently inserted between peripheral and central devices and acts like a transparent proxy breaking up the peer-to-peer model. It maintains BLE link layer connections to peripheral devices within its range. A central controller implements functionalities such as service discovery, access policy enforcement, and subscription management so that features like service slicing, enrichment, and composition can be realized by BLESS.

Muppet [403] extends BLESS by supporting the Zigbee protocol in parallel to BLE. In addition to the features of BLESS, inter-protocol services between Zigbee and BLE and BLE/Zigbee and IP protocols are introduced. An example for the latter are HTTP transactions that are automatically sent out by the switch if it sees a specified set of BLE/Zigbee transactions. The data plane implementation of BLESS is extended by protocol-dependent packet parsers and processing and support for encrypted Zigbee packets via packet recirculation.

Wang et al. [404] implement aggregation and disaggregation of small IoT packets on P4 switches. For a small IoT packet, the header holds a large proportion of the packet's total size. In large streams of IoT packets, this causes high overhead. The current aggregation techniques for IoT packets are implemented by external servers or on the control plane of switches, both resulting in low throughput and added latency. Therefore, the authors propose an implementation directly on P4 switches where IoT packets are buffered, aggregated, and encapsulated in UDP packets with a custom flag-header, type, and padding. In disaggregation, the incoming packet is cloned to stripe out the single messages until all messages are separated.

Madureira et al. [405] present the *Internet of Things Protocol (IoTP)*, an L2 communication protocol for IoT data planes. The main purpose of IoTP is data aggregation at the network level. IoTP introduces a new, fixed header and is compatible with any forwarding mechanism. The authors implemented IoTP for the bmv2 and store single packets of a flow in registers until the data can be aggregated.

Engelhard et al. [406] present a system for massive wireless sensor networks. They implement a physically distributed, and logically centralized wireless access systems to reduce the impairment by collisions. P4 is leveraged as connection between a physical access point and a virtual access point. To that end, they extend the bmv2 to provide additional functionality. However, they give information about their P4 program only in form of a decision flow graph.

12.3. Industrial Networking

FastReact [407] outsources sensor data packet processing from centralized controllers to P4 switches. The sensor data is recorded in variable-length time

series data stores where an additional field holds the current moving average calculated on the time series. Both data for all sensors can be polled by a central controller. For controlling actuators directly on the data plane, FastReact supports the formulation of control logic in conjunctive normal form (CNF). It is mapped to actions to either forward signal data to the controller, discard it, or directly send it to the actuator. FastReact also features failure recovery directly on the switch. For every sensor and actuator, timestamps for the last received packets along a timeout limit is recorded. If failures are detected, sensor data are forwarded following failover rules with backup actuators for particular sensors.

Cesen et al. [408] leverage P4-capable switches to move control logic to the network. Control applications reside in controllers that are responsible for emergency intervention, e.g., if a given threshold is exceeded. The connection to the controller may be faulty and, therefore, controller intervention may not be fast enough. In this work, the authors generate emergency packets, i.e., stop commands, directly in the data plane. The action is triggered if the switch receives a packet with a specific payload.

Kunze et al. [409] investigate the applicability of in-network computing to industrial environments. They offload a simple task, i.e., coordinate transformation, to different programmable P4 targets. They come to the conclusion, that, while in general possible, even simple task have heavy demands on programmable network devices and that offloading may lead to inaccurate results.

12.4. Time-Sensitive Networking (TSN)

Rüth et al. [410] introduce a scheme for implementing in-network control mechanisms for linear quadratic regulators (LQR). LQRs can be described by a multiplication of a matrix and a vector. The vector describes the control of the actuator, the matrix describes the current system state. The result of the multiplication is a control command. The destination of a switch describes a specific actuator. When a switch receives a control packet, it matches the destination of the packet onto a match-and-action table. The lookup provides the control vector for the actuator. The control vector from the lookup is then multiplied with the system state matrix that is stored in a register to calculate the control command for the actuator. The resulting control command is written into the packet header and the packet is forwarded to the target actuator.

Kannan et al. [411] introduce the Data Plane Time synchronization Protocol (DPTP) for distributed applications with computations directly on the P4 data plane. DPTP follows a request-response model, i.e., all P4 switches request the global time from a designated master switch. Therefore, each switch features a local control plane that generates time requests sent to the master switch. Additionally, the control plane handles overflows in time calculation for administration.

Kundel et al. [412] demonstrate timestamping with nanosecond accuracy. They describe a simple setup with a Tofino-based switch and a breakout cable to connect two ports of the switch. In the experiment, timestamps at the moment

of sending and reception are recorded in the packet header. The authors compare those two timestamps to show that very fine-grained measurements are possible.

12.5. Network Function Virtualization (NFV)

Kathará [413] runs NFs as P4 programs either on software or hardware targets. For software-based deployment, the framework leverages Docker containers that run NFs as container images or individual setups for Quagga, Open vSwitch, or bmv2 container images. For hardware-based deployment on P4 switches, NFs are either replicated on every P4 switch or distributed on multiple P4 switches as needed. In both cases, a load balancer or service classifier forwards flows to the appropriate P4 switch. As a main advantage, P4 programs can be shifted between the bmv2-based P4 software targets and hardware targets depending on the required performance.

P4NFV [414] also deals with the idea of running NFs either on software- or hardware-based P4 targets. The authors adopt the ETSI NFV architecture with control and monitoring entities and add a layer that abstracts various types of software- and hardware-based P4 targets as P4 nodes. For optimized deployment, the targets performance characteristics are part of the P4 node description. For runtime reconfiguration, the authors propose two approaches. In pipeline manipulation, the P4 program features multiple match-action pipelines that can be enabled or disabled by setting register flags. In program reload, a new P4 program is compiled and loaded to the P4 target. The authors propose to perform state management and migration either directly on the data plane or via a control plane.

Osiński et al. [415] use P4 to offload the data plane of virtual network functions (VNFs) into a cloud infrastructure by allowing VNFs to inject small P4 programs into P4 devices like SmartNICs or top-of-rack switches. This results in better performance and a microservice-based approach for the data plane. A new P4 architecture model that integrates abstractions used to develop VNF data planes was developed.

Moro et al. [416] present a framework for NF decomposition and deployment. They split NFs into components that can run on CPUs or that can be offloaded to specific programmable hardware, e.g., P4 programmable switches. The presented orchestrator combines multiple functions into a single P4 program that can be deployed to programmable switches.

DPPx [417] implements a framework for P4-based data plane programmability and exposure which allows enhancing NFV services. They introduce data plane modules written in P4 which can be leveraged by the application plane. As an example, a dynamic optimization of packet flow routing (DOPFR) is implemented using DPPx.

Mohammadkhan et al. [418] provide a unified P4 switch abstraction framework where servers with software NFs and P4-capable SmartNICs are seen as one logical entity by the SDN controller. They further leverage Mixed Integer Linear Programming (MILP) to determine partitioning of P4 tables for optimal placement of NFs.

FOP4 [419] [420] implements a rapid prototyping platform that supports container-based, P4-switch-based, and SmartNIC-based NFs. They argue that a prototyping platform is needed to quickly develop and evaluate new NFV use cases.

PlaFFE [421] introduces NFV offloading where some features of VNFs or embedded Network Functions (eNFs) are executed on SmartNICs using P4. Additionally, P4 is used to steer traffic either through the eNFs or through VNFs using SR-IOV.

12.6. Service Function Chains (SFCs)

P4SC [422] [423] implements a SFC framework for P4 targets. SFCs are described as directed acyclic graph of service functions (SFs). In P4SC, SFs are represented by blocks. Each block has a unique identifier, a P4 program for ingress processing, and a P4 program for egress processing. P4SC includes 15 SF blocks, e.g., L2 forwarding, which are extracted from switch.p4. After the user specified all SFCs for a particular P4 target, the P4SC converter merges the directed acyclic graphs of all SFCs with an LCS-based algorithm into an intermediate representation. Then, the P4SC generator creates the final P4 program based on the intermediate representation to be deployed onto the P4 target. P4 program generation includes runtime management, i.e., the generator creates one API per SFC while hiding SF-specific details, e.g., names of particular match-and-action tables.

Re-SFC [425] improves P4SC's resource usage by using resubmit operations. If the specified order of SFs in an SFC does not match the pre-embedded SF of the P4 switch, incoming flows cannot be processed. P4SC solves this problem by permitting redundant NF embeds, i.e., if SFs of one SFC are required by another SFCs, those SFs are just replicated. To reduce the costly usage of match-and-action tables, Re-SFC introduces resubmit actions where packets are re-bounced to the ingress.

FlexMesh [426] tackles the problem of fixed SFC flow control, i.e., when the specified order of SFs does not match the pre-embedded SF, by leveraging MATs. SFs can be dynamically bypassed, and recirculation is used to build any desired SF chain.

P4-SFC [427] is an SFC framework based on MPLS segment routing and NFV. P4 is used to implement a traffic classifier. A central orchestrator deploys service functions as VNFs and configures the traffic classifier based on definitions of SFCs.

12.7. Summary and Analysis

As the research domain of advanced networking covers different topics, almost all core properties of P4 are covered. The area of cellular networks (Section 12.1) greatly benefits from the *definition and usage of custom packet headers* as many works are based on tunneling technologies, such as GTP. Further, *flexible packet header processing* allows implementing new 5G concepts such as gNB or EPG. Some use cases still require offloading tasks to specialized hardware

or software by leveraging the *target-specific packet header processing function* property of P4, e.g., for ARQ or ciphering in the context of gNB. Network function virtualization (NFV) (Section 12.5) benefits from *flexible development and deployment* as single network functions (NFs) can be replaced or relocated during operation. New protocols and extensions to existing protocols presented in Section 12.6 rely on *definition and usage of custom packet headers* and *flexible packet header processing.*

13. Applied Research Domains: Network Security

We describe applied research on firewalls, port knocking, DDoS attack mitigation, intrusion detection systems, connection security, and other fields of application. Table 10 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

13.1. Firewalls

Ricart-Sanchez et al. [429] present a 5G firewall that analyzes GTP data transmitted between edge and core networks. P4 allows an implementation of parsing and matching GTP header fields such as 5G user source IP, 5G user destination IP, and identification number of the GTP tunnel. The P4 pipeline implements an allow-by-default policy, DROP actions for specific sets of keys can be installed via a data plane API. In a follow-up work [430], the authors extend the 5G firewall by support for multi-tenancy with VXLAN.

CoFilter [431] implements an efficient flow identification scheme for stateful firewalls in P4. To solve the problem of limited table sizes on SDN switches, flow identifiers are calculated by applying a hashing function to the 5-tuple of every packet directly on the switch. The proposed concept includes a novel hash rewrite function that is implemented on the data plane. It resolves hash commission and hash table optimization using an external server.

P4Guard [432] replaces software-based firewalls by P4-based virtual firewalls in the VNGuard [480] system. VNGuard introduces controller-based deployment and management of virtual firewalls with the help of SDN and NFV. The P4-based firewall comprises a single MAT that allows ALLOW/DROP decision for Layer 3/4 header fields as match keys. The flow statistics are recorded with the help of counters. Another MAT allows enabling/disabling the firewall at runtime.

Vörös and Kiss [433] present a firewall implemented in P4. The parser supports Ethernet, IPv4/IPv6, UDP, and TCP headers. A ban list comprises MAC address/IP address entries that represent network hosts. Packets matching this ban list are directly dropped. To mitigate port scan or DDoS attacks, counters track packet rate and byte transfer statistics. Another MAT implements whitelist filtering.

Table 10: Overview of applied research on network security (Section 13).

Research work	Year	Targets	Code
Firewalls (13.1)			
Ricart-Sanchez et al. [429, 430]	2018/19	NetFPGA-SUME	
CoFilter [431]	2018	Tofino	
P4Guard [432]	2018	bmv2	
Vörös and Kiss [433]	2016	p4c-behavioral	
Port Knocking (13.2)			
P4Knocking [434]	2020	bmv2	
Almaini et al. [435]	2019	bmv2	
DDoS Mitigation Mechanisms (13.3)			
LAMP [436]	2018	bmv2	
TDoSD@DP [437, 438]	2018/19	bmv2	
Kuka et al. [439]	2019	Xilinx UltraScale+, Intel Stratix 10	
Paolucci et al. [440, 441]	2018/19	bmv2, NetFPGA-SUME	
ML-Pushback [442]	2019	-	
Afek et al. [443]	2017	p4c-behavioral	
Cardoso Lapolli et al. [444]	2019	bmv2	[445]
Cai et al. [446]	2020	-	
Lin et al. [447]	2020	bmv2	
Musumeci et al. [448]	2020	bmv2	
DIDA [449]	2020	bmv2	
Dimolianis et al. [450]	2020	Netronome	
Scholz et al. [451]	2020	bmv2, T_4P_4S, Netronome, NetFPGA SUME	[452]
Friday et al. [453]	2020	bmv2	
NetHide [454]	2018	-	
Intrusion Detection Systems & Deep Packet Inspection (13.4)			
P4ID [455]	2019	bmv2	
Kabasele and Sadre [456]	2018	bmv2	
DeepMatch [457]	2020	Netronome	[458]
Qin et al. [459]	2020	bmv2, Netronome	[460]
SPID [461]	2020	bmv2	

Research work	Year	Targets	Code
Other Fields of Application (13.6)			
Chang et al. [462]	2019	bmv2	
Clé [463]	2019	-	
P4DAD [464]	2020	bmv2	
Chen [465]	2020	Tofino	[466]
Gondaliya et al. [467]	2020	NetFPGA SUME	
Poise [468]	2020	Tofino	[469]
Connection Security (13.5)			
P4-MACsec [470]	2020	bmv2, NetFPGA-SUME	[471]
P4-IPsec [472]	2020	bmv2, NetFPGA-SUME, Tofino	[473]
SPINE [474]	2019	bmv2	[475]
Qin et al. [476]	2020	bmv2	
P4NIS [477]	2020	bmv2	[478]
LANIM [479]	2020	bmv2	

13.2. Port Knocking

Port knocking is a simple authentication mechanism for opening network ports. Network hosts send TCP SYN packets in predefined sequences to certain ports. If the sequence is completed correctly, the server opens up a desired port. Typically, port knocking is implemented in software on servers.

P4Knocking [434] implements port knocking on P4 switches. The authors propose four different implementations for P4. In the first implementation, P4 switches track the state of knock sequences in registers where the source IP address is used as an index. The second implementation uses a CRC-hash of the source IP address as index for the knocking state registers. To resolve the problem of hash collisions, the third implementation relies on identifiers that are calculated and managed by the controller. The fourth implementation solely relies on the controller, i.e., P4 switches forward all knocking packets to the controller.

Almaini et al. [435] implement port knocking with a ticket mechanism on P4 switches. Traffic is only forwarded if the sender has a valid ticket. Predefined trusted nodes have a ticket by default, untrustworthy nodes must obtain a ticket by successful authentication via port knocking. The authors use the HIT/MISS construct of P4 as well as stateful P4 components to implement the concept. Port knocking sequences and trusted/untrusted hosts can be maintained by the control plane.

13.3. DDoS Attack Mitigation

LAMP [436] presents a cooperative mitigation mechanism for DDoS attacks that relies on information from the application layer. Ingress P4 switches add

a unique identifier to the IP options header field of any processed packet. The last P4 switch ahead of the target host stores this mapping and empties the IP options header field. If a network hosts, e.g., a database server, detects an ongoing DDoS attack on the application layer, it adds an attack flag to the IP options header field and sends it back to the switch. The switch forwards this packet to the ingress switch to enable dropping of all further packets of this flow.

TDoSD@DP [437] is a P4-based mitigation mechanism for DDoS attacks targeting SIP proxies. Stateful P4 registers record the number of SIP INVITE and SIP BYE messages. Then, a simple state machine monitors sequences of INVITE and BYE messages. Many INVITES followed by zero BYE messages lead to dropping SIP INVITE packets where valid sequences of INVITE and BYE messages will keep the port open. In a follow-up work [438], the authors present an alternative approach where P4 switches act as distributed sensors. An SDN controller periodically collects data from counters of P4 switches to perform centralized attack detection. Then, attack mitigation is performed by installing DROP rules on the P4 switches.

Kuka et al. [439] present a DDoS mitigation system that targets volumetric DDoS attacks called reflective amplification attacks. The authors port an existing VHDL implementation into a P4 program that runs on FPGA targets. The implementation selects the affected subset of the incoming traffic, extracts packet data, and forwards it as a digest to an SDN controller. The SDN controller continuously evaluates this information; a heuristic algorithm identifies aggressive IP addresses by looking at the volumetric contribution of source IP addresses to the attack. In case of a detected attack, the SDN controller installs DROP rules.

Paolucci et al. [440, 441] present a stateful mitigation mechanism for TCP SYN flood attacks. It is part of a P4-based edge packet-over-optical node that also comprises traffic engineering functionality. P4 registers keep per-session statistics to detect TCP SYN flood attacks. One register records the port number of the last TCP SYN packet, the another one records the number of attempts matching the TCP SYN flood behavior. If the latter one exceeds a defined threshold, the packets are dropped.

ML-Pushback [442] proposes an extension of the Pushback DDoS attack mitigation mechanism by machine learning techniques. P4 switches implement a data collector mechanism that collects dropped packets and forwards them as digest messages to the control plane. On the control plane, a deep learning module extracts signatures and classifies the collected digest with a decision tree model. Attack mitigation is performed by throttling attacker traffic via rate limits.

Afek et al. [443] implement known mitigation mechanisms for SYN and DNS spoofing in DDoS attacks for OpenFlow and P4 targets. The OpenFlow implementation targets Open vSwitch and OpenFlow 1.5 where P4 implementations are compiled for p4c-behavioral without control plane involvement. In addition, the authors implemented a set of algorithms and methods for dynamically distributing the rule space over multiple switches.

Cardoso Lapolli et al. [444] describe an algorithmic approach to detect and stop DDoS attacks on P4 data planes. The algorithm was specifically created under the functional constraints of P4 and is based on the calculation of the Shannon entropy.

Cai et al. [446] propose a novel method for collecting traffic information to detect TCP port scanning attacks. The authors propose the "0-replacement" method as an efficient alternative to existing sampling and aggregation methods. It introduces a pending request counter (PRcounter) and relies on registers to bind hashing identifiers of the attackers' IP addresses to PRcounter values. The authors describe the concept as compliant to PSA, but only simulation results are given.

Lin et al. [447] present a comparison of OF- and P4-based implementations of basic mitigation mechanisms against SYN flooding and ARP spoofing attacks.

Musumeci et al. [448] present P4-assisted DDoS attack mitigation using an ML classifier. An ML-based DDoS attack detection module with a classifier is running on a controller. The P4 switch forwards traffic to the module; the DDoS attack detection module responds with a decision. The authors consider three use cases: packet mirroring + header mirroring + metadata extraction. In metadata extraction, P4 switches implement counters that store occurrences of IP, UDP, TCP, and SYN packets. In the case that one of the counters exceeds a defined threshold, the P4 switch inserts a custom header with the counter values and sends it to the DDoS attack detection module.

DIDA [449] presents a distributed mitigation mechanism against amplified reflection DDoS attacks. In this type of DDoS attack, spoofed requests lead to responses that are by magnitude larger. An example is a DNS ANY query. The authors rely on count-min sketch data structures and monitoring intervals to put the number of requests and responses into relation. In case of a detected DDoS attack, ACLs are used to block the traffic near to the attacker.

Dimolianis et al. [450] introduce a multi-feature DDoS detection scheme for TCP/UDP traffic. It considers the total number of incoming traffic for a particular network, the significance of the network, and the symmetry ratio of incoming and outgoing traffic for classifications. The feature analysis is time-dependent and focuses on distinct time intervals.

Scholz et al. [451] propose a SYN proxy that relies on SYN cookies or SYN authentication as protection against SYN flooding DDoS attacks. The authors present a software implementation based on DPDK and compare it to a bmv2-based P4 implementation that is ported to the T_4P_4S P4 software target, Netronome P4 hardware target, and NetFPGA SUME P4 hardware target. Evaluation results, benefits, and challenges for each platform are discussed.

Friday et al. [453] present a two-part DDoS detection and mitigation scheme. In the first part, a P4 target applies a one-way traffic analysis using bloom filters and time-dependent statistics such as moving averages. In the second part, the P4 target analyzes the bandwidth and transport protocols used by various applications to perform a volumetric analysis. The processing pipeline then decides about malicious traffic to be dropped. Administrators may supply custom network parameters used for dynamic threshold calculation that are

then installed via an API on the data plane. The authors demonstrate the effectiveness of the proposed approach by three use cases: UDP amplification DDoS attacks, SYN flooding DDoS attacks, and slow DDoS attacks.

NetHide [454] prevents link-flooding attacks by obfuscating the topology of a network. It achieves this by modifying path tracing probes in the data plane while preserving their usability.

13.4. Intrusion Detection Systems (IDS) & Deep Packet Inspection (DPI)

P4ID [455] reduces intrusion detection system (IDS) processing load by apply pre-filtering on P4 switches (IDS offloading/bypassing). P4ID features a rule parser that translates Snort rules with a multistage mechanism into MAT entries. The P4 processing pipeline implements a stateless and a stateful stage. In the stateless stage, TCP/ICMP/UDP packets are matched against a MAT to decide if traffic should be dropped, forwarded to the next hop, or forwarded to the IDS. In the stateful stage, the first n packets of new flows are forwarded to the IDS. This allows that traffic targeting well-known ports can be also analyzed. Combining the feedback of the IDS for packet samples with the stateless stage is future work.

Kabasele and Sadre [456] present a two-level IDS for industrial control system (ICS) networks. The IDS targets the Modbus protocol that runs on top of TCP in SCADA networks. The first level comprises two whitelists: a flow whitelist for filtering on the TCP layer and a Modbus whitelist. If no matching entry is found for a given packet, it is forwarded to the second layer. This is in stark contrast to legacy whitelisting where packets are just dropped. In the second level, a Zeek network security analyzer acts as deep packet inspector running on a dedicated host. It analyzes the given packet, makes a decision, and instructs the controller to update filters on the switch.

DeepMatch [457] introduces deep packet inspection (DPI) for packet payloads. The concept is implemented with the help of network processors; its prototype is built with the Netronome NFP-6000 SmartNIC P4 target. The authors present regex matching capabilities that are executed in 40 Gbit/s (line rate of the platform) for stateless intra-packet matching and about 20 Gbit/s for stateful inter-packet matching. The DeepMatch functionalities are natively implemented in Micro-C for the Netronome platform and integrated into the P4 processing pipeline with the help of P4 externs.

Qin et al. [459] present an IDS based on binarized neural networks (BNN) and federated learning. BNNs compress neural networks into a simplified form that can be implemented on P4 data planes. Weights are compressed into single bits and computations, e.g., activation functions, are converted into bit-wise operations. P4 targets at the network edge then apply BNNs to classify incoming packets. To continuously train the BNNs on the P4 targets, the authors propose a federated learning scheme. Each P4 target is connected to a controller that trains an equally structured neural network with samples received from the P4 target. A cloud service aggregates local updates received from the controllers and responds with weight updates that are processed into the local model.

In the Switch-Powered Intrusion Detection (SPID) framework [461], switches compute and store flow statistics, and perform traffic change detection. If a relevant change in traffic is detected, measurement data is pushed to the control plane. In the control plane, the measurement data is fed to a ML-based anomaly detection pipeline to detect potential attacks.

13.5. Connection Security

P4-MACsec [470] presents an implementation of IEEE 802.1AE (MACsec) for P4 switches. A two-tier control plane with local switch controllers and a central controller monitor the network topology and automatically set up MACsec on detected links between P4 switches. For link discovery and monitoring, the authors implement a secured variant of LLDP that relies on encrypted payloads and sequence numbers. MACsec is directly implemented on the P4 data plane; encryption/decryption using AES-GCM is implemented on the P4 target and integrated in the P4 processing pipeline as P4 externs.

P4-IPsec [472] presents an implementation of IPsec for P4 switches. IPsec functionality is implemented in P4 and includes ESP in tunnel mode with support for different cipher suites. As in P4-MACsec, the cipher suites are implemented on the P4 target and integrated as P4 externs. In contrast to standard IPsec operation, IPsec tunnels are set up and renewed by an SDN controller without IKE. Site-to-site operation mode supports IPsec tunnels between P4 switches. Host-to-site operation mode supports roadwarrior access to an internal network via a P4 switch. To make the roadwarrior host manageable by the controller, the authors introduce a client agent tool for Linux hosts.

SPINE [474] introduces surveillance protection in the network elements by IP address obfuscation against surveillance in intermediate networks. In contrast to software-based approaches such as TOR, SPINE runs entirely on the data plane of two nodes with intermediate networks in between. It applies a one-time-pad-based encryption scheme with key rotation to encrypt IP addresses and, if present, TCP sequence and acknowledgment numbers. The SPINE nodes add a version number representing the encryption key index to each packet by which the receiving switch can select the appropriate key for decryption. The key sets required for the key rotation are maintained by a central controller.

Qin et al. [476] introduce encryption of TCP sequence numbers using substitution-boxes to protect traffic between two P4 switches. An ONOS-based controller receives the first packet of each new flow and applies security policies to decide whether the protection should be enabled. Then, it installs the necessary data in registers and updates MATs to enable TCP sequence number substitution.

P4NIS [477] proposes a scheme to protect against eavesdropping attacks. It comprises three lines of defense. In the first line of defense, packets that belong to one traffic flow are disorderly transmitted via various links. In the second line of defense, source/destination ports and sequence/acknowledgment numbers are substituted via s-boxes similar to the approach of Qin et al. [476]. The third line of defense resembles existing encryption mechanisms that are not covered by P4NIS.

LANIM [479] presents a learning-based adaptive network immune mechanism to prevent against eavesdropping attacks. It targets the Smart Identifier Network (SINET) [481], a novel, three-layer Internet architecture. LANIM applies the minimum risk ML algorithm to respond to irregular conditions and applies a policy-based encryption strategy focusing on the intent and application.

13.6. Other Fields of Application

Chang et al. [462] present IP source address encryption. It accomplishes non-linkability of IP addresses as proactive defense mechanism. Network hosts are connected to trusted P4 switches at the network edges. In between, packets are exchanged via untrusted switches/routers. The P4 switch next to the sender encrypts the sender IP address by applying an XOR operation with a hash calculated by a random number and a shared key. The P4 switch next to the receiver decrypts the original sender IP address. The mechanism includes a dynamic key update mechanism so that transformations are random.

Clé [463] proposes to upgrade particular switches in a legacy network to P4 switches that implement security network functions (SNFs) such as rule-based firewalls or IDS on P4 switches. Clé comprises a smart device upgrade selection algorithm that selects switches to be upgraded and a controller that forwards traffic streams to the P4 switches that implement SNFs.

P4DAD [464] presents a novel approach to secure duplicate address detection (DAD) against spoofing attacks. Duplicate address detection is part of NDP in IPv6 where nodes check if an IPv6 address to be applied conflicts with another node. As the messages exchanged in duplicate address detection are not authenticated or encrypted, it is vulnerable to message spoofing. As simple alternative to authentication or encryption, P4DAD introduces a mechanism to filter spoofed NDP messages. The P4 switch maintains registers to create bindings between IPv6 addresses, port numbers, and address states. Thereby, it can detect and drop spoofed NDP messages.

Chen [465] shows how AES can be implemented on Tofino-based P4 targets in P4 using MATs as lookup tables. Expansion of the AES key is performed in the control plane. MAT entries specific to the encryption keys are generated by a controller.

Gondaliya et al. [467] implement six known mechanisms against IP address spoofing for the NetFPGA SUME P4 target. Those are Network Ingress Filtering, Reverse Path Forwarding (Loose, Strict and Feasible), Spoofing Prevention Method (SPM), and Source Address Validation Improvement (SAVI). The authors compare the different mechanisms with regard to resource usage on the FPGA and report that the implementations of all mechanisms achieve a throughput of about 8.5 Gbit/s and a processing latency of about 2 µs per packet.

Poise [468] introduces context-aware policies for securing P4-based networks in BYOD scenarios. Instead of relying on a remote controller or software-based solution, Poise implements context-aware policy enforcement directly on P4 tar-

gets. Network administrators define context-aware security policies in a declarative language based on Pyretic NetCore that are then compiled into P4 programs to be executed on P4 targets. BYOD clients run a context collection module that adds context information headers to network packets. The P4 program generated by Poise then parses and uses this information to enforce ACLs based on device runtime contexts. P4 targets in Poise are managed by a Poise controller that compiles the P4 programs, installs them on the P4 targets, and provides configuration data to the collection modules. The authors present a prototype including PoiseDroid, an implementation of the context collection module for Android devices.

13.7. Summary and Analysis

Several prototypes apply P4's *custom packet headers*, e.g., for building a GTP firewall for 5G networks, a DDoS attack mitigation mechanism for the SIP, or an IDS for the Modbus protocol in industrial networks. It is also used for in-band signaling, e.g., in cooperative DDoS attack detection. All prototypes rely on *flexible packet header processing*; outstanding for this section, many of them also rely on *target-specific packet header processing functions* offered by the P4 target. Some works require custom externs, e.g., for applying MACsec or IPsec on P4 data planes. As for prototypes from the research area *Monitoring* (Section 9), many prototypes rely on registers and counters for recording statistics, e.g., for detecting attacks in DDoS mitigation or in IDSs. While custom packet headers and basic packet header processing are supported by all P4 hardware targets, the portability of prototypes using these specific functions is very limited. Several prototypes also rely on *packet processing on the control plane* where information (e.g., from blocking lists, IDS rules) is translated into MAT rules for data plane control or data received from the data plane (e.g., statistical data or packet digests) is used for runtime control. *Flexible deployment* allows to re-deploy network security programs on P4 switches in large networks.

14. Miscellaneous Applied Research Domains

This section summarizes work that falls outside of the other application domains. We describe applied research on network coding, distributed algorithms, state migration, and application support. Table 11 shows an overview of all the work described. At the end of the section, we summarize the work and analyze it with regard to P4's core features described in Section 8.1.

14.1. Network Coding

In Network Coding (NC) [521], linear encoding and decoding operations are applied on packets to increase throughput, efficiency, scalability, and resilience. Network nodes apply primitive operations, e.g., splitting, encoding, or decoding packets, to implement NC mechanisms such as multicast, forward error correction, or rerouting (resilience).

Table 11: Overview of applied research on miscellaneous research domains (Section 14).

Research work	Year	Targets	Code
Network Coding (Section 14.1)			
Kumar et al. [482]	2018	bmv2	[483]
Gonçalves et al. [484]	2019	bmv2	
Distributed Algorithm (Section 14.2)			
P4CEP [485]	2018	bmv2, Netronome	
DAIET [486]	2017	-	
Sankaran et al. [487]	2020	-	
Zang et al. [488]	2017	bmv2	
Dang et al. [489, 490]	2016/20	Tofino	[491]
P4BFT [492, 493]	2019	bmv2, Netronome	
SwiShmem [494]	2020	-	
SC-BFT [495]	2020	bmv2	[496]
LODGE [497]	2018	bmv2	
LOADER [498]	2020		[499]
FLAIR [500]	2020	Tofino	
State Migration (Section 14.3)			
Swing State [501]	2017	bmv2	
P4Sync [502]	2020	bmv2	[503]
Xue et al. [504]	2020	bmv2	
Kurzniar et al. [505]	2020	bmv2	
Sankaran et al. [506]	2020	NetFPGA-SUME	
Application Support (Section 14.4)			
P4DNS [507]	2019	NetFPGA SUME	[508]
P4-BNG [509]	2019	bmv2, Tofino, Netronome, NetFPGA-SUME	[510]
ARP-P4 [511]	2018	bmv2	
Glebke et al. [512]	2019	Netronome	
COIN [513]	2019	-	
Lu et al. [514]	2019	Tofino	
Yazdinejad et al. [515]	2019	bmv2	
P4rt-OVS [516]	2020	-	[517]
SwitchML [518]	2021	Tofino	[519]
SwitchAgg [520]	2019	NetFPGA-SUME	

Kumar et al. [482] implement primitive NC operations such as splitting, encoding, and decoding for a PSA software switch. This is the first introduction of NC for SDN, as fixed-function data plane switches, e.g., as in OF, did not support such operations. The authors describe details of their implementation. The open-source implementation [483] relies on clone and recirculate operations to generate additional packets for encoding and decoding operations and packet processing loops. Temporary packet buffers for gathering operations are implemented with P4 registers. However, P4 hardware targets are not considered.

Gonçalves et al. [484] implement NC operations that may use information from multiple packets during processing. The authors implement their concept for PISA in $P4_{16}$. It features multiple complex NC operations that focus on multiplications in Galois fields used for encoding and decoding operations. NC operations are implemented in P4 externs that extend the capabilities of the software switch to store a specific amount of received packets. Again, hardware targets are not considered.

14.2. Distributed Algorithms

We describe related work on event processing and in-network consensus.

14.2.1. Event Processing

Data with stream characteristics often require specific processing. For example, sensor data may be analyzed to determine whether values are within certain thresholds, or chunks of data are aggregated and preprocessed.

P4CEP [485] shifts complex event processing from servers to P4 switches so that event stream data, e.g., from sensors, is directly processed on the data plane. The solution requires several workarounds to solve P4 limitations regarding stateful packet processing.

DAIET [486] introduces in-network data aggregation where the aggregation task is offloaded to the entire network. This reduces the amount of traffic and reliefs the destination of computational load. The authors provide a prototype implementation in $P4_{14}$ but only a few details are disclosed.

Sankaran et al. [487] increase the processing speed of packets by reducing the time that is required by forwarding nodes to parse the packet header. To that end, ingress routers parse the header stack to compute a so-called unique parser code (UPC) which they add to the packet header. Downstream nodes need to parse only the UPC to make forwarding decisions.

14.2.2. In-Network Consensus

Distributed algorithms or mechanisms may require consensus to determine the right solution or processing. This includes communication between participating entities and some ways to determine the right solution.

Zhang et al. [488] propose to offload parts of the Raft consensus algorithm to P4 switches. However, the mechanisms require an additional client to run on the switch. The authors implement their application for a P4 software switch, but details are not presented.

Dang et al. [489, 490] describe a P4 implementation of Paxos, a protocol that solves consensus for distributed algorithms in a network of unreliable processors based on information exchange between switches. This work contains a detailed description of a complex P4 implementation. The authors explain all components, provide code snippets, and discuss their design choices.

P4BFT [492, 493] introduces a consensus mechanism against buggy or malicious control plane instances. The controller responses are sent to trustworthy instances which compare the responses and establish consensus, e.g., by choosing the most common response. The authors propose to offload the comparison process to the data plane.

SwiShmem [494] is a distributed shared state management layer for the P4 data plane to implement stateful distributed network functions. In high-performance environments controllers are easily overloaded when consistency of write-intensive distributed network functions, like DDoS detection, or rate limiters, is required. Therefore, SwiShmem offloads consistency mechanisms from the control plane to the data plane. Then, consistency mechanisms operate at line rate because switches process traffic, and generate and forward state update messages without controller interaction.

Byzantine fault refers to a system where consensus between multiple entities has to be established where one or more entities are unreliable. Byzantine fault tolerance (BFT) describes mechanisms that handle such faults. However, BFTs often require significant time to reach consensus due to high computational overhead to reduce uncertainty. Switch-centric BFT (SC-BFT) [495] proposes to offload BFT functionalities, i.e., time synchronization and state synchronization, into the data plane. This significantly accelerates the consensus procedure since nodes process information at line rate.

LODGE [497] implements a mechanism for switches to make forwarding decisions based on global state without control of a central instance. Developers define global state variables which are stored by all stateful data plane devices. When such a node processes a packet that changes a global state variable, the switch generates and forwards an update packet to all other stateful switches on a predefined distribution tree. LOADER [498] introduces global state to the data plane. Consensus is maintained by the data plane devices through distributed algorithms, i.e., the switches send notification messages when global state changes. This increases scalability in comparison to mechanisms where consensus is managed by a central control entity.

FLAIR [500] accelerates read operations in leader-based consensus protocols by processing the read requests in the data plane. To that end, FLAIR devices in the core maintain persistent information about pending write operations on all objects in the system. When a client submits a read request, the FLAIR switch checks whether the requested object is stable, i.e., if it has pending write operations. If the object is stable, the FLAIR switch instructs another client with a stable version of the object, to send it to the requesting client. If the object is not stable, the FLAIR switch forwards the write request to the leader.

14.3. State Migration

In Swing State [501], switches maintain state in registers that should be migrated to other nodes. For migration, state information is carried by regular packets created by the P4 clone operation throughout the network.

P4Sync [502] is a protocol to migrate data plane state between switches. Thereby, it does not require controller interaction and provides guarantees on the authenticity of the transferred state. To that end, it leverages the switch's packet generator to transfer the content of registers between devices. Authenticity in a migration operation is guaranteed by a hash chain where each packet contains the hashed values of both the current payload and the payload of the previous packet.

Xue et al. [504] propose a hybrid approach for storing flow entries to address the issue of limited on-switch memory. While some flow entries are still stored in the internal memory of the switch, some flow entries may be stored on servers. Switches access them with only low latency via remote direct memory access (RDMA).

Kuzniar et al. [505] propose to leverage programmable switches to act as in-network cache to speed up queries over encrypted data stores. Encrypted key-value pairs are thereby stored in registers.

Sankaran et al. [506] describe a system to relieve switches from parsing headers. They propose to parse headers at an ingress switch only and add a *unique parser code* to the packet that identifies the set of headers of the packet. With this information, following switches can parse relevant information from the headers without having to parse the whole header stack.

14.4. Application Support

This subsection describes work that focuses on support or implementation of existing applications and protocols.

P4DNS [507] is an in-network DNS system. The authors propose a hybrid architecture with performance-critical components in the data plane and components with flexibility requirements in the control plane. The data plane responds to DNS requests and forwards regular traffic while cache management, recursive DNS requests, and uncached DNS responses are handled by the control plane.

P4-BNG [509] implements a carrier-grade broadband network gateway (BNG) in P4. The authors aim to provide an implementation for many different targets. To that end, they introduce a layer between data plane and control plane. This hardware-specific BNG data plane controller runs directly on the targets to provide a uniform interface to the control plane. It then configures the data plane according to the control commands from the control plane.

ARP-P4 [511] implements MAC address learning based on ARP solely on the P4 data plane. To substitute a control plane, the authors integrate MAC learning as an external function.

Glebke et al. [512] propose to offload computer vision functionalities, in particular, time-critical computations, to the data plane. To that end, the

authors leverage convolution filters on a P4-programmable NIC. The necessary computations are distributed to various MATs.

COordinate-based INdexing (COIN) [513] is a mechanism to ensure efficient access to data on multiple distributed edge servers. To that end, the authors introduce a centralized instance that indexes data and its associated location. When an edge server requires data that it has not cached itself, it requests the data index at the centralized instance which provides a data location.

Lu et al. [514] propose intra-network inference (INI) and implement it in P4. It offloads neural network computations into the data plane. To that end, each P4 switch communicates via USB with a dedicated neural compute stick which performs computations.

Yazdinejad et al. [515] present a P4-based blockchain enabled packet parser. The proposed architecture focuses on FPGAs and aims to bring the security characteristics of blockchains into the data plane to greatly increase processing speed.

P4rt-OVS [516] is an extension for the OVS based on BPFs to combine the programmability of P4 and the well-known features of the OVS. P4rt-OVS enables runtime programming of the OVS, in particular, the deployment of new network features without recompilation of the OVS. It contains a P4-to-BPF compiler which allows developers to write data plane code for the OVS in P4.

SwitchML [518] proposes to accelerate distributed machine learning training with programmable switch data planes. Within distributed machine learning, so-called worker nodes compute model updates on a subset of the training data. Afterwards, these model updates are synchronized and merged on the worker nodes. The authors of SwitchML design a communication primitive to perform parts of the model aggregation within the network. They evaluate their algorithm on the Tofino platform and show an increase in training performance up to a factor of 5.5.

SwitchAgg [520] proposes a switch design for in-network aggregation that solves shortcomings of common reprogrammable switches. It processes packets at line rate and drastically reduces the required network traffic for distributed algorithms. The authors implement and evaluate their switch design in Verilog HDL on a NetFPGA-SUME.

14.5. Summary and Analysis

P4 facilitates the development of prototypes in the domain of network coding (see Subection 14.1) by providing *target-specific packet header processing functions*. The prototypes heavily rely on externs to implement complex packet processing behavior, i.e., encoding and decoding operations, packet splitting and packet merging. Such prototypes were mainly developed for the bmv2 and portability to hardware platforms depends on the properties of the used externs and the capabilities of the hardware targets. Distributed algorithms (see Section 14.2) leverage all sorts of P4's core features. Some prototypes *define and use custom packet headers* to transport information that are not available in standard protocols. Others rely on *flexible packet header processing* and

target-specific packet header processing functions to implement unconventional and complex packet processing behavior. Some prototypes require *packet processing on the control plane* to resolve consistency issues or make network-wide configuration decisions. In the context of state migration (see Section 14.3) the prototypes mainly leverage externs to enable stateful processing. As a result, most projects were developed for the bmv2 with only limited portability to hardware platforms. Finally, some prototypes reimplement traditional network protocols or network elements, e.g., DNS, BNG, or ARP. Those projects mainly *define and use custom packet headers* for information transport, *flexible packet header processing* to implement the functionality of the specific protocol or network element, *target-specific packet header processing functions* for complex packet processing, and *packet processing on the control plane* for corner cases.

15. Discussion & Outlook

We discuss the findings of this survey and present an outlook.

15.1. P4 as a Language for Programmable Data Planes

From a variety of data plane programming approaches, P4 became the currently most widespread standard. Learning resources (Section 3.8) and the bmv2 P4 software target (Section 5.1) constitute low entry barriers for P4 technology. This is appealing for academia, and hardware support on high-speed platforms make P4 relevant for industry. The large body of literature that we surveyed in this work demonstrates that P4 has the right abstractions to build prototypes for many use cases in different application domains. Moreover, P4 allows simple and flexible definition of data plane APIs (Section 6) that can be used by simple control plane programs or complex, enterprise-grade SDN controllers. Thus, P4 allows practitioners and researchers to express their data plane and control plane algorithms in a simple way and thereby unleashes a great innovation potential. As P4 is supported by multiple platforms, there is a potentially large user group. In addition, P4 is an open programming language so that the source code can be published as open source. Therefore, public P4 code can profit from a large user community, both in quantity and quality, which is a benefit for software maintenance and security.

We consider P4 as a milestone technology. It offers great flexibility and an easy, generalized, yet powerful abstraction to describe data plane behavior. Its main objective is high-speed packet header processing. Its wide support by high-speed hardware targets enables prototype development for many different use cases.

15.2. P4 Targets Revisited

We have listed many available P4 targets in Section 5. However, our literature overview showed that mostly the bmv2 development and testing platform

as well as P4 hardware targets based on the Tofino ASIC were applied in the reviewed papers.

The vast majority of prototypes runs on the software switch bmv2. One reason is that it is freely available for everyone. In addition, the complexity of the code is not constrained by hardware restrictions. And finally, any required extern can be customized. Therefore, there is no limit on algorithmic complexity so that bmv2 can serve as a platform for any use case – but only from a functional point of view. As it is a pure software-based prototyping solution, it cannot provide high throughput and is, therefore, not suitable for deployment in production environments.

The Intel Tofino family of Intelligent Fabric Processor (IFP) ASICs is currently the most popular hardware target and the only programmable data plane platform with throughput rates up to 25.6 Tbit/s and ports running at up to 400 Gbit/s, making it appropriate for production environments like data centers or core networks. Tofino uses P4 as native programming language. Therefore, comprehensive tools are offered to support the P4 development process on this platform. Moreover, P4 gives access to all features of the Tofino chip so that there is no penalty of using P4 as a programming language. Existing restrictions are due to the functional limitation of a high-speed platform. Thus, prototypes for Tofino are more challenging but prove the technical feasibility of a new concept at commercial scale. Probably for these reasons the Tofino turned out to be the mostly used hardware platform in our survey.

P4 can be also used on FPGA- or NPU-based targets. They come with only a few ports and lower throughput rates so that they may be used for special-purpose server applications but not for typical switching devices. They excel through the possibility to extend the target functionality with user-defined externs. These cards are typically programmed by vendor-specific languages. P4 support is achieved by trans-compilers that translate P4 programs into the vendor-specific format. P4 programmability might be limited to a restricted feature set while access to all features of a target is only possible through the vendor-specific programming language. Whether the application of P4 for such targets is beneficial compared to vendor-specific programming languages or interfaces, mainly depends on the use case, level of knowledge of the programmer, and if prospect target-independence is a goal.

15.3. Portability, Target- and Vendor-Independence

Portability is an important advantage of using a high-level programming language such as P4. While the subject of portability is explicitly discussed in the $P4_{16}$ specification (see Section 3), practical implications are frequently misunderstood. In general, P4 programs are not expected to be portable across different P4 architectures. P4 programs written for a given P4 architecture should be portable across all P4 targets that implement the corresponding model, provided there are sufficient resources on the P4 target. Even if two P4 targets support the same P4 architecture, a P4 program written for one P4 target might not compile to the other P4 target because of the differences in the available resources. The only portability guarantee that is made is that if the program can

be successfully compiled on both P4 targets, it will exhibit the same behavior and produce the same results. This guarantee is somewhat weaker, compared to what portability means in the general purpose programming languages.

There have been several efforts to define portable P4 architectures. For network switches, it is mainly the Portable Switch Architecture (PSA). Their main challenge is not in the language, but the capabilities of existing high-speed hardware. While software P4 targets such as the bmv2 have no difficulties implementing any P4 architecture, it is almost impossible to emulate a non-existing capability or provide an adaptation layer on a high-speed hardware P4 target; simply because of the lack of sufficient resources. This is especially true for any differences that can be found in fixed-function components and externs. As a result, today's efforts tend to codify the "lowest common denominator" functionality that is guaranteed to be found on multiple P4 targets while carefully avoiding codifying any behavior that might differ. This severely limits the ability of P4 programs to fully use the capabilities of the chosen P4 targets and thus almost all P4 code surveyed today tends to use native P4 architectures instead.

Portable P4 architectures still do not provide target- or vendor-independence, i.e., the ability to simply recompile a P4 program without any changes on a different P4 target for either the same or a different vendor. This is due to the fact that the availability of specific resources differs among P4 targets.

We evaluated the specific P4 targets chosen by the authors of the surveyed works. Thereby, we noticed several important trends. First, the majority of works have been implemented either for the bmv2 P4 target, P4 targets with the Tofino ASIC, or both. When both implementations were present, the authors tend to keep their implementations for the bmv2 P4 target and Tofino P4 target separate as two independent P4 programs. Quite often, the implementations are highly different and many authors had dedicated sections in their works explaining the required major changes in porting a P4 program written for the bmv2 P4 target into a P4 program for Tofino P4 targets. A number of authors specifically mention that they could implement their P4 program only on some targets but not on others. Reasons are specific hardware resource limits, e.g., number of stages, and hardware constraints, e.g., available operations and number of operations per packet. They are naturally present on all high-speed targets. Additional reasons are special externs and fixed-function functionality that are only available on specific P4 targets.

15.4. A Business Perspective for P4-Programmable Data Planes

Today, the most prevalent hardware network appliances are proprietary devices for which customized hardware and software are jointly developed.

Data plane programming breaks with this process. Programmable packet processing ASICs such as the Tofino may be sold by specialized manufacturers and integrated by other vendors with a motherboard, CPU, memory, and connectors in white box switches. The accompanying software, i.e., data plane and control plane programs, might be provided by the same vendor, a third party, or implemented by the users themselves.

Because software is developed independently of hardware, the agility of the development process can be increased, which can reduce the time to market. Hardware platforms become reusable; they can be leveraged for multiple purposes with the help of appropriate P4 programs.

Network solution providers may leverage the lowered entry barrier for customized hardware appliances to develop and sell P4 software for various P4-capable targets, at least with moderate adaptation effort. A decade of implementation experience may no longer be a prerequisite for that business.

In addition, companies with large networks and particular use cases, e.g., special applications in data centers, may use customized algorithms to overcome inefficiencies of standardized protocols or mechanisms.

Large companies can avoid vendor lock-in by acquisition of programmable components instead of black boxes. The components are assembled possibly with open-source software leveraging data plane programming, SDN, and NFV. The ACCESS 4.0 architecture [522] and the O-RAN Alliance [523] are examples. This type of disaggregation also enables cost scaling effects where off-the-shelf components are bought at moderate cost instead of expensive specialized appliances.

15.5. Outlook

P4 is a programming language for a diverse set of programmable network targets. Currently, its main practical application are high-speed switches. It is supported by Intel's Tofino ASIC, but other manufacturers like Xilinx and Pensando recently also launched P4-based products.

The many prototypes surveyed in this paper showed that there is a need for more functionality on programmable switches, which may be provided by extern functions. While they reduce portability, they enable more use cases. Examples for such extern functions are features that have been used in some of the pure software-based P4 prototypes. They encrypt and decrypt packet payload, support floating-point operations, provide flexible hash functions, or allow more complex calculations. Those externs might be provided by the target manufacturers for common use cases or integrated by users.

Hardware with a vendor-specific programming language may benefit from offering interfaces and cross-compilers for P4 together with useful extern functions. Although this may not give access to the full functionality of the platform, users with P4 programming knowledge can customize such devices for their needs without worrying about hardware details.

The biggest driver for P4 is possibly disaggregation. While currently devices from different vendors can be orchestrated by a customized controller, P4 may have the potential to extend disaggregation towards specialized appliances based on off-the-shelf programmable hardware. Hardware without an open programming interface cannot profit from that market.

16. Conclusion

In this paper, we first gave a tutorial on data plane programming with P4. We delineated it from SDN and introduced programming models with a special focus on PISA which is most relevant for P4. We provided an overview of the current state of P4 with regard to programming language, architectures, compilers, targets, and data plane APIs. We reported research efforts to advance P4 that fall in the areas of optimization of development and deployment, research on P4 targets, and P4-specific approaches for control plane operation.

In the second part of the paper, we analyzed 245 papers on applied research that leverage P4 for implementation purposes. We categorized these publications into research domains, summarized their key points, and characterized them by prototype, target platform, and source code availability. For each research domain, we presented an analysis on how works benefit from P4. To that end, we identified a small set of core features that facilitate implementations. The survey proved a tremendous uptake of P4 for prototyping in academic research from 2018 to 2021. One reason is certainly the multitude of openly available resources on P4 and the bmv2 P4 software target. They are an ideal starting point for creating P4-based prototypes, even for beginners.

The many P4-based activities which emerged only within short time show that P4 technology can speed up the evolution of computer networking. While multiple hardware targets are available, most hardware-based prototypes leverage the Tofino ASIC that is optimized for high throughput on many ports and particularly suited for data center and WAN applications. However, the majority of P4-based prototypes was implemented with the bmv2 software switch. Many of them were not ported to hardware, probably due to the complexity of their data plane algorithms and lack of required extern functions on current hardware. This may change in the future if new P4 hardware targets are available. We expect P4 to become a base technology for multiple hardware appliances, in particular in the context of disaggregation and for small-scale markets.

17. Acknowledgement

This work was partly supported by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of this paper.

References

[1] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, The Click Modular Router, ACM Transactions on Computer Systems (TOCS) 18 (2000) 217–231.

[2] VPP/What is VPP?, `https://bit.ly/2mrxVGE`, accessed 01-20-2021 (2021).

[3] GitHub: NPL-Spec, https://github.com/nplang/NPL-Spec, accessed 01-20-2021 (2021).

[4] Software Defined Specification Environment for Networking (SDNet), https://www.xilinx.com/support/documentation/backgrounders/sdnet-backgrounder.pdf, accessed 01-20-2021 (2021).

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming Protocol-independent Packet Processors, ACM SIGCOMM Computer Communications Review (CCR) 44 (2014) 87–95.

[6] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 1617–1634.

[7] Y. Jarraya, T. Madi, M. Debbabi, A Survey and a Layered Taxonomy of Software-Defined Networking, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 1955–1980.

[8] W. Xia, Y. Wen, C. H. Foh, D. Niyato, H. Xie, A Survey on Software-Defined Networking, IEEE Communications Surveys & Tutorials (COMST) 17 (2015) 27–51.

[9] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, M. Nogueira, Programmable Networks—From Software-Defined Radio to Software-Defined Networking, IEEE Communications Surveys & Tutorials (COMST) 17 (2015) 1102–1125.

[10] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE 103 (2015) 14–76.

[11] R. Masoudi, A. Ghaffari, Software defined networks: A survey, Journal of Network and Computer Applications (JNCA) 67 (2016) 1–25.

[12] C. Trois, M. D. Del Fabro, L. C. E. de Bona, M. Martinello, A Survey on SDN Programming Languages: Toward a Taxonomy, IEEE Communications Surveys & Tutorials (COMST) 18 (2016) 2687–2712.

[13] W. Braun, M. Menth, Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices, MDPI Future Internet Journal (FI) 6 (2014) 302–336.

[14] F. Hu, Q. Hao, K. Bao, A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 2181–2206.

[15] A. Lara, A. Kolasani, B. Ramamurthy, Network Innovation using OpenFlow: A Survey, IEEE Communications Surveys & Tutorials (COMST) 16 (2014) 493–512.

[16] R. Bifulco, G. Rétvári, A Survey on the Programmable Data Plane: Abstractions, Architectures, and Open Problems, in: IEEE International Conference on High Performance Switching and Routing (HPSR), 2018, pp. 1–7.

[17] E. Kaljic, A. Maric, P. Njemcevic, M. Hadzialic, A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking, IEEE ACCESS 7 (2019) 47804–47840.

[18] O. Michel, R. Bifulco, G. Rétvári, S. Schmid, The Programmable Data Plane: Abstractions, Architectures, Algorithms, and Applications, ACM Computing Surveys 1 (2021).

[19] S. Kaur, K. Kumar, N. Aggarwal, A review on p4-programmable data planes: Architecture, research efforts, and future directions, Computer Communications 170 (2021).

[20] E. F. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, ArXiv e-prints (2021).

[21] Y. Gao, Z. Wang, A Review of P4 Programmable Data Planes for Network Security, Mobile Information Systems (2021).

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Computer Communications Review (CCR) 38 (2008) 69–74.

[23] BESS: Berkeley Extensible Software Switch, `http://span.cs.berkeley.edu/bess.html`, accessed 01-20-2021 (2021).

[24] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN, ACM SIGCOMM Conference 43 (2013) 99–110.

[25] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy, A. Orda, T. Edsall, DRMT: Disaggregated Programmable Switching, in: ACM SIGCOMM Conference, 2017, p. 1–14.

[26] Google Presentations: P4 Tutorial, `http://bit.ly/p4d2-2018-spring`, accessed 01-20-2021 (2018).

[27] Website of the P4 Language Consortium, https://p4.org/, accessed 01-20-2021 (2021).

[28] The P4 Language Specification, https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf, accessed 01-20-2021 (2018).

[29] P4 16 Language Specification (v.1.2.1, https://p4.org/p4-spec/docs/P4-16-v1.2.1.html, accessed 01-20-2021 (2020).

[30] M. Moshref, A. Bhargava, A. Gupta, M. Yu, R. Govindan, Flow-level State Transition as a New Switch Primitive for SDN, in: ACM SIGCOMM Conference, 2014, p. 61–66.

[31] G. Bianchi, M. Bonola, A. Capone, C. Cascone, OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch, ACM SIGCOMM Computer Communications Review (CCR) 44 (2014) 44–51.

[32] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, S. Licking, Packet Transactions: High-Level Programming for Line-Rate Switches, in: ACM SIGCOMM Conference, 2016, p. 15–28.

[33] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici, G. Siracusano, FlowBlaze: Stateful Packet Processing in Hardware, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2019, p. 531–547.

[34] H. Song, Protocol-Oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding Plane, in: ACM Workshop on Hot Topics in Networks (HotNets), 2013, p. 127–132.

[35] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, D. Walker, NetKAT: Semantic Foundations for Networks, in: ACM Symposium on Principles of Programming Languages (POPL), 2014, p. 113–126.

[36] P4 Tutorial, https://github.com/p4lang/tutorials, accessed 05-05-2021 (2021).

[37] P4 Guide, https://github.com/jafingerhut/p4-guide, accessed 05-05-2021 (2021).

[38] P4 Learning, https://github.com/nsg-ethz/p4-learning, accessed 05-05-2021 (2021).

[39] Charter of the P4 Architecture WG, https://github.com/p4lang/p4-spec/blob/master/p4-16/psa/charter/P4_Arch_Charter.mdk, accessed 01-20-2021 (2021).

[40] P4_16 PSA Specification (v1.1), https://p4lang.github.io/p4-spec/docs/PSA-v1.1.0.html, accessed 01-20-2021 (2018).

[41] P4-HLIR Specification v.0.9.30, https://github.com/p4lang/p4-hlir/blob/master/HLIRSpec.pdf, accessed 01-20-2021 (2016).

[42] GitHub: p4c, https://github.com/p4lang/p4c, accessed 01-20-2021 (2021).

[43] P. G. Patra, C. E. Rothenberg, G. Pongracz, MACSAD: High Performance Dataplane Applications on the Move, in: IEEE International Conference on High Performance Switching and Routing (HPSR), 2017, pp. 1–6.

[44] Open Data Plane, https://opendataplane.org/, accessed 01-20-2021 (2021).

[45] L. Jose, M. R. N. M. Lisa Yan, Stanford University; George Varghese, Compiling Packet Programs to Reconfigurable Switches, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2015, p. 103–115.

[46] P. Li, Y. Luo, P4GPU: Accelerate Packet Processing of a P4 Program with a CPU-GPU Heterogeneous Architecture, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2016, pp. 125–126.

[47] GitHub: p4c-behavioural, https://github.com/p4lang/p4c-behavioral/tree/master/p4c_bm, accessed 01-20-2021 (2021).

[48] GitHub: Behavioural Model Version 2 (BMv2), https://github.com/p4lang/behavioral-model, accessed 01-20-2021 (2021).

[49] P4 Behaviour Model: Why did we need BMv2, https://github.com/p4lang/behavioral-model\#why-did-we-replace-p4c-behavioral-with-bmv2, accessed 01-20-2021 (2021).

[50] GitHub: Behavioral model targets, https://github.com/p4lang/behavioral-model/blob/master/targets/README.md, accessed 01-20-2021 (2021).

[51] BMv2 Performance, https://github.com/p4lang/behavioral-model/blob/master/docs/performance.md, accessed 01-20-2021 (2021).

[52] GitHub: eBPF Backend for p4c, https://github.com/p4lang/p4c/tree/master/backends/ebpf, accessed 01-20-2021 (2021).

[53] p4c-ubpf: a New Back-end for the P4 Compiler, https://p4.org/p4/p4c-ubpf.html, accessed 01-20-2021 (2021).

[54] GitHub: p4c-xdp, https://github.com/vmware/p4c-xdp, accessed 01-20-2021 (2021).

[55] P4@ELTE, http://p4.elte.hu/, accessed 01-20-2021 (2021).

[56] S. Laki, D. Horpácsi, P. Vörös, R. Kitlei, D. Leskó, M. Tejfel, High speed packet forwarding compiled from protocol independent data plane specifications, in: ACM SIGCOMM Conference, 2016, p. 629–630.

[57] Data Plane Development Kit (DPDK), https://www.dpdk.org/, accessed 01-20-2021 (2021).

[58] GitHub: T4P4S, https://github.com/P4ELTE/t4p4s, accessed 01-20-2021 (2021).

[59] A. Bhardwaj, A. Shree, V. B. Reddy, S. Bansal, A Preliminary Performance Model for Optimizing Software Packet Processing Pipelines, in: ACM SIGOPS Asia-Pacific Workshop on System (APSys), 2017, pp. 1–7.

[60] X. Wu, P. Li, T. Miskell, L. Wang, Y. Luo, X. Jiang, Ripple: An Efficient Runtime Reconfigurable P4 Data Plane for Multicore Systems, in: International Conference on Networking and Network Applications (NaNA), 2019, pp. 142–148.

[61] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, J. Rexford, PISCES: A Programmable, Protocol-Independent Software Switch, in: ACM SIGCOMM Conference, 2016, p. 525–538.

[62] Open vSwitch, https://www.openvswitch.org/, accessed 01-20-2021 (2021).

[63] GitHub: PISCES, https://github.com/P4-vSwitch, accessed 01-20-2021 (2021).

[64] S. Choi, X. Long, M. Shahbaz, S. Booth, A. Keep, J. Marshall, C. Kim, The Case for a Flexible Low-Level Backend for Software Data Planes, in: Asia-Pacific Workshop on Networking (APnet), 2017, p. 71–77.

[65] S. Choi, X. Long, M. Shahbaz, S. Booth, A. Keep, J. Marshall, C. Kim, PVPP: A Programmable Vector Packet Processor, in: ACM Symposium on SDN Research (SOSR), 2017, p. 197–198.

[66] Northbound Networks - Who are You?, https://northboundnetworks.com/pages/about-us, accessed 01-20-2021 (2021).

[67] GitHub: ZodiacFX-P4, https://github.com/NorthboundNetworks/ZodiacFX-P4, accessed 01-20-2021 (2021).

[68] GitHub: p4c-zodiacfx, https://github.com/NorthboundNetworks/p4c-zodiacfx, accessed 01-20-2021 (2021).

[69] P. Zanna, P. Radcliffe, K. G. Chavez, A Method for Comparing OpenFlow and P4, in: International Telecommunication Networks and Applications Conference (ITNAC), 2019, pp. 1–3.

[70] GitHub: P4-NetFPGA, https://github.com/NetFPGA/P4-NetFPGA-public/wiki, accessed 01-20-2021 (2021).

[71] S. Ibanez, G. Brebner, N. McKeown, N. Zilberman, The P4-NetFPGA Workflow for Line-Rate Packet Processing, in: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2019, p. 1–9.

[72] N. Zilberman, Y. Audzevich, G. A. Covington, A. W. Moore, NetFPGA SUME: Toward 100 Gbps as Research Commodity, IEEE Micro 34 (2014) 32–41.

[73] Netcope P4, https://www.netcope.com/Netcope/media/content/NetcopeP4_2019_web.pdf, accessed 01-20-2021 (2021).

[74] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, H. Weatherspoon, P4FPGA: A Rapid Prototyping Framework for P4, in: ACM Symposium on SDN Research (SOSR), 2017, p. 122–135.

[75] GitHub: P4FPGA, https://github.com/p4fpga/p4fpga, accessed 01-20-2021 (2021).

[76] P. Benácek, V. Pu, H. Kubátová, P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers, in: IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2016, pp. 148–155.

[77] P. Benáček, V. Puš, J. Kořenek, M. Kekely, Line Rate Programmable Packet Processing in 100Gb Networks, in: International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1–1.

[78] J. Cabal, P. Benáček, L. Kekely, M. Kekely, V. Puš, J. Kořenek, Configurable FPGA Packet Parser for Terabit Networks with Guaranteed Wire-Speed Throughput, in: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2018, p. 249–258.

[79] S. da Silva, Jeferson, Boyer, François-Raymond, Langlois, J. Pierre, P4-Compatible High-Level Synthesis of Low Latency 100 Gb/s Streaming Packet Parsers in FPGAs, in: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2018, p. 147–152.

[80] M. Kekely, J. Korenek, Mapping of P4 Match Action Tables to FPGA, in: International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1–2.

[81] R. Iša, P. Benáček, V. Puš, Verification of Generated RTL from P4 Source Code, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 444–445.

[82] Z. Cao, H. Su, Q. Yang, J. Shen, M. Wen, C. Zhang, P4 to FPGA-A Fast Approach for Generating Efficient Network Processors, IEEE ACCESS 8 (2020) 23440–23456.

[83] Z. Cao, H. Su, Q. Yang, M. Wen, C. Zhang, A Template-based Framework for Generating Network Processor in FPGA, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 1057–1058.

[84] Open Tofino, https://github.com/barefootnetworks/open-tofino, accessed 01-22-2021 (2021).

[85] EdgeCore Wedge 100BF-32X, https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335, accessed 01-20-2021 (2021).

[86] APS Networks BF2556X-1T-A1F, https://stordirect.com/shop/switches/25g-switches/aps-networks-bf2556x-1t-a1f/, acessed 01-22-2021 (2021).

[87] APS Networks BF6064X-T-A2F, https://stordirect.com/shop/switches/100g-switches/aps-networks-bf6064x-t-a2f/, acessed 01-22-2021 (2021).

[88] Netberg Aurora 610, https://netbergtw.com/products/aurora-610/, accessed 01-20-2021 (2021).

[89] Arista Press Release: Arista Announces New Multi-function Platform for Cloud Networking, https://www.arista.com/en/company/news/press-release/5148-pr-20180605, accessed 01-20-2021 (2021).

[90] Cisco Blog: Increase Flexibility with Cisco's Programmable Cloud Infrastructure, https://blogs.cisco.com/datacenter/increase-flexibility-with-ciscos-programmable-cloud-infrastructure, accessed 01-20-2021 (2021).

[91] SONiC - Supported Platforms, https://azure.github.io/SONiC/Supported-Devices-and-Platforms.html, accessed 01-20-2021 (2021).

[92] A. Seibulescu, M. Baldi, Leveraging P4 Flexibility to Expose Target-Specific Features, in: P4 Workshop in Europe (EuroP4), 2020, p. 36–42.

[93] The Pensando Distributed Services Platform, https://pensando.io/our-platform/, accessed 01-20-2021 (2021).

[94] Netronome: P4 Data Plane Programming, https://netronome.com/media/documents/WP_P4_Data_Plane_Programming.pdf, accessed 01-20-2021 (2018).

[95] Netronome: Programming with P4 and C, https://www.netronome.com/media/documents/WP_Programming_with_P4_and_C.pdf, accessed 09-20-2019 (2018).

[96] H. Harkous, M. Jarschel, M. He, R. Pries, W. Kellerer, Towards Understanding the Performance of P4 Programmable Hardware, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–6.

[97] Apache Thrift, https://thrift.apache.org/, accessed 01-20-2021 (2021).

[98] gRPC, https://grpc.io/, accessed 01-20-2021 (2021).

[99] Google Protocol Buffers, https://developers.google.com/protocol-buffers/, accessed 01-20-2021 (2021).

[100] Charter of the P4 API WG, https://github.com/p4lang/p4-spec/blob/master/api/charter/P4_API_WG_charter.mdk, accessed 01-20-2021 (2021).

[101] P4 Runtime API Specification v.1.3.0 (2019-12-01), https://p4.org/p4runtime/spec/v1.3.0/P4Runtime-Spec.html, accessed 01-20-2021 (2020).

[102] ONOS: P4 brigade, https://wiki.onosproject.org/display/ONOS/P4+brigade, accessed 01-20-2021 (2021).

[103] OpenDaylight: P4 brigade, P4PluginDeveloperGuide, accessed 09-23-2019 (2019).

[104] B. O'Connor, Y. Tseng, M. Pudelko, C. Cascone, A. Endurthi, Y. Wang, A. Ghaffarkhah, D. Gopalpur, T. Everman, T. Madejski, J. Wanderer, A. Vahdat, Using P4 on Fixed-Pipeline and Programmable Stratum Switches, in: P4 Workshop in Europe (EuroP4), 2010, pp. 1–2.

[105] GitHub: P4tutorial, https://github.com/p4lang/tutorials/tree/master/utils/p4runtime_lib, accessed 01-20-2021 (2021).

[106] GitHub: PI Library, https://github.com/p4lang/PI, accessed 01-20-2021 (2021).

[107] GitHub: Behavioural Model - simple_switch_grpc, https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc, accessed 01-20-2021 (2021).

[108] GitHub: bmv2 Runtime CLI, https://github.com/p4lang/behavioral-model/blob/master/tools/runtime_CLI.py, accessed 01-20-2021 (2021).

[109] E. O. Zaballa, Z. Zhou, Graph-to-P4: A P4 Boilerplate Code Generator for Parse Graphs, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–2.

[110] Y. Zhou, J. Bi, ClickP4: Towards Modular Programming of P4, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 100–102.

[111] M. Baldi, daPIPE A Data Plane Incremental Programming Environment, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[112] M. Eichholz, E. Campbell, N. Foster, G. Salvaneschi, M. Mezini, How to Avoid Making a Billion-Dollar Mistake: Type-Safe Data Plane Programming with SafeP4, in: European Conference on Object-Oriented Programming (ECOOP), 2019, pp. 1–28.

[113] M. Riftadi, F. Kuipers, P4I/O: Intent-Based Networking with P4, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 438–443.

[114] L. Yu, J. Sonchack, V. Liu, Mantis: Reactive Programmable Switches, in: ACM SIGCOMM Conference, 2020, p. 296–309.

[115] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, M. Yu, Lyra: A Cross-Platform Language and Compiler for Data PlaneProgramming on Heterogeneous ASICs, in: ACM SIGCOMM Conference, 2020, p. 435–450.

[116] M. Riftadi, J. Oostenbrink, F. Kuipers, GP4P4: Enabling Self-Programming Networks, ArXiv e-prints (2019).

[117] D. Moro, D. Sanvito, A. Capone, FlowBlaze.p4: a library for quick prototyping of stateful SDN applications in P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 95–99.

[118] D. Moro, D. Sanvito, A. Capone, Demonstrating FlowBlaze.p4: fast prototyping for EFSM-based data plane applications, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 116–117.

[119] D. Moro, D. Sanvito, A. Capone, Developing EFSM-Based Stateful Applications with FlowBlaze.P4 and ONOS, in: P4 Workshop in Europe (EuroP4), 2020, p. 52–53.

[120] N. Sultana, J. Sonchack, H. Giesen, I. Pedisich, Z. Han, N. Shyamkumar, S. Burad, A. DeHon, B. T. Loo, Flightplan: Dataplane disaggregation and placement for p4 programs, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2021, pp. 571–592.

[121] R. Shah, A. Shirke, A. Trehan, M. Vutukuru, P. Kulkarni, pcube: Primitives for Network Data Plane Programming, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 430–435.

[122] Z. Ma, J. Bi, C. Zhang, Y. Zhou, A. B. Dogar, CacheP4: A Behavior-level Caching Mechanism for P4, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 108–110.

[123] A. Abhashkumar, J. Lee, J. Tourrilhes, S. Banerjee, W. Wu, J.-M. Kang, A. Akella, P5: Policy-driven Optimization of P4 Pipeline, in: ACM Symposium on SDN Research (SOSR), 2017, p. 136–142.

[124] P. Wintermeyer, M. Apostolaki, A. Dietmüller, L. Vanbever, P2GO: P4 Profile-Guided Optimizations, in: ACM Workshop on Hot Topics in Networks (HotNets), 2020, p. 146–152.

[125] S. Yang, L. Baia, L. Cui, Z. Ming, Y. Wu, S. Yu, H. Shen, Y. Pan, P4 Edge node enabling stateful traffic engineering and cyber security, Journal of Network and Computer Applications (JNCA) 171 (2020) A84–A95.

[126] B. Vass, E. Bérczi-Kovács, C. Raiciu, G. Rétvári, Compiling Packet Programs to Reconfigurable Switches: Theory and Algorithms, in: P4 Workshop in Europe (EuroP4), 2020, p. 28–35.

[127] S. Abdi, U. Aftab, G. Bailey, B. Boughzala, F. Dewal, S. Parsazad, E. Tremblay, PFPSim: A Programmable Forwarding Plane Simulator, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2016, pp. 55–60.

[128] J. Bai, J. Bi, P. Kuang, C. Fan, Y. Zhou, C. Zhang, NS4: Enabling Programmable Data Plane Simulation, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–7.

[129] C. Fan, J. Bi, Y. Zhou, C. Zhang, H. Yu, NS4: A P4-Driven Network Simulator, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 105–107.

[130] N. McKeown, D. Talayco, G. Varghese, N. P. Lopes, N. Bjørner, A. Rybalchenko, Automatically Verifying Reachability and Well-Formedness in P4 Networks, `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/09/p4nod.pdf`, accessed 01-20-2021 (2016).

[131] A. Kheradmand, G. Rosu, P4K: A Formal Semantics of P4 and Applications, ArXiv e-prints (2018).

[132] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, N. Foster, P4V: Practical Verification for Programmable Data Planes, in: ACM SIGCOMM Conference, 2018, p. 490–503.

[133] L. Freire, M. Neves, L. Leal, K. Levchenko, A. Schaeffer-Filho, M. Barcellos, Uncovering Bugs in P4 Programs with Assertion-based Verification, in: ACM Symposium on SDN Research (SOSR), 2018, p. 1–7.

[134] M. Neves, L. Freire, A. Schaeffer-Filho, M. Barcellos, Verification of P4 Programs in Feasible Time using Assertions, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2018, p. 73–85.

[135] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, C. Raiciu, Debugging P4 Programs with Vera, in: ACM SIGCOMM Conference, 2018, p. 518–532.

[136] M. A. Noureddine, A. Hsu, M. Caesar, F. A. Zaraket, W. H. Sanders, P4AIG: Circuit-Level Verification of P4 Programs, in: IEEE/IFIP International Conference on Dependable Systems and Networks – Supplemental Volume (DSN-S), 2019, pp. 21–22.

[137] D. Dumitrescu, R. Stoenescu, L. Negreanu, C. Raiciu, bf4: towards bug-free P4 programs, in: ACM SIGCOMM Conference, 2020, p. 571–585.

[138] D. Dumitrescu, R. Stoenescu, M. Popovici, L. Negreanu, C. Raiciu, Dataplane equivalence and its applications, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2019, pp. 683–698.

[139] F. Yousefi, A. Abhashkumar, K. Subramanian, K. Hans, S. Ghorbani, A. Akella, Liveness Verification of Stateful Network Functions, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 257–272.

[140] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, P. Athanas, P4Pktgen: Automated Test Case Generation for P4 Programs, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–7.

[141] Y. Zhou, J. Bi, Y. Lin, Y. Wang, D. Zhang, Z. Xi, J. Cao, C. Sun, P4Tester: Efficient Runtime Rule Fault Detection for Programmable Data Planes, in: IEEE International Workshop on Quality of Service (IWQoS), 2019, pp. 1–10.

[142] GitHub: P4app, `https://github.com/p4lang/p4app`, accessed 01-20-2021 (2021).

[143] A. Shukla, K. N. Hudemann, A. Hecker, S. Schmid, Runtime Verification of P4 Switches with Reinforcement Learning, in: Workshop on Network Meets AI & ML, 2019, p. 1–7.

[144] D. Jindal, R. Joshi, B. Leong, P4TrafficTool: Automated Code Generation for P4 Traffic Generators and Analyzers, in: ACM Symposium on SDN Research (SOSR), 2019, p. 152–153.

[145] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, H. Weatherspoon, Whippersnapper: A P4 Language Benchmark Suite, in: ACM Symposium on SDN Research (SOSR), 2017, p. 95–101.

[146] F. Rodriguez, P. G. K. Patra, L. Csikor, C. E. Rothenberg, P. Vörös, S. Laki, G. Pongrácz, BB-Gen: A Packet Crafter for P4 Target Evaluation, in: ACM SIGCOMM Conference Posters and Demos, 2018, p. 111–113.

[147] H. Harkous, M. Jarschel, M. He, R. Pries, W. Kellerer, P8: P4 with Predictable Packet Processing Performance, IEEE Transactions on Network and Service Management (TNSM) (2020) 1–1.

[148] S. Kodeswaran, M. T. Arashloo, P. Tammana, J. Rexford, Tracking P4 Program Execution in the Data Plane, in: ACM Symposium on SDN Research (SOSR), 2020, p. 117–122.

[149] K. Birnfeld, D. C. da Silva, W. Cordeiro, B. B. N. de França, P4 Switch Code Data Flow Analysis: Towards Stronger Verification of Forwarding Plane Software, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–8.

[150] M. Neves, B. Huffaker, K. Levchenko, M. Barcellos, Dynamic Property Enforcement in Programmable Data Planes, in: IFIP-TC6 Networking Conference (Networking), 2019, pp. 1–9.

[151] C. Zhang, J. Bi, Y. Zhou, J. Wu, B. Liu, Z. Li, A. B. Dogar, Y. Wang, P4DB: On-the-fly Debugging of the Programmable Data Plane, in: IEEE International Conference on Network Protocols (ICNP), 2017, pp. 1–10.

[152] Y. Zhou, J. Bi, C. Zhang, B. Liu, Z. Li, Y. Wang, M. Yu, P4DB: On-the-Fly Debugging for Programmable Data Planes, IEEE/ACM Transactions on Networking (ToN) 27 (2019) 1714–1727.

[153] M. Neves, K. Levchenko, M. Barcellos, Sandboxing Data Plane Programs for Fun and Profit, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 103–104.

[154] A. Shukla, S. Fathalli, T. Zinner, A. Hecker, S. Schmid, P4Consist: Toward Consistent P4 SDNs, IEEE Journal on Selected Areas in Communications (JSAC) 38 (2020) 1293–1307.

[155] Z. Xia, J. Bi, Y. Zhou, C. Zhang, KeySight: A Scalable Troubleshooting Platform Based on Network Telemetry, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–2.

[156] F. Ruffy, T. Wang, A. Sivaraman, Gauntlet: Finding Bugs in Compilers for Programmable Packet Processing, in: USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020, pp. 1–17.

[157] J. Krude, J. Hofmann, M. Eichholz, K. Wehrle, A. Koch, M. Mezini, Online Reprogrammable Multi Tenant Switches, in: ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, 2019, p. 1–8.

[158] D. Hancock, J. van der Merwe, HyPer4: Using P4 to Virtualize the Programmable Data Plane, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2016, p. 35–49.

[159] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, J. Wu, HyperV: A High Performance Hypervisor for Virtualization of the Programmable Data Plane, in: IEEE International Conference on Computer Communications and Networks (ICCCN), 2017, pp. 1–9.

[160] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, J. Wu, MPVisor: A Modular Programmable Data Plane Hypervisor, in: ACM Symposium on SDN Research (SOSR), 2017, p. 179–180.

[161] GitHub: HyperVDP, `https://github.com/HyperVDP`, accessed 01-20-2021 (2021).

[162] C. Zhang, J. Bi, Y. Zhou, J. Wu, HyperVDP: High-Performance Virtualization of the Programmable Data Plane, IEEE Journal on Selected Areas in Communications (JSAC) 37 (2019) 556–569.

[163] M. Saquetti, G. Bueno, W. Cordeiro, J. R. Azambuja, P4VBox: Enabling P4-Based Switch Virtualization, IEEE Communications Letters 24 (2020) 146–149.

[164] M. Saquetti, G. Bueno, W. Cordeiro, J. R. Azambuja, VirtP4: An Architecture for P4 Virtualization, in: IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2019, pp. 75–78.

[165] P. Zheng, T. Benson, C. Hu, P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2018, p. 98–111.

[166] R. Parizotto, L. Castanheira, F. Bonetti, A. Santos, A. Schaeffer-Filho, PRIME: Programming In-Network Modular Extensions, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[167] E. O. Zaballa, D. Franco, M. S. Berger, M. Higuero, A Perspective on P4-Based Data and Control Plane Modularity for Network Automation, in: P4 Workshop in Europe (EuroP4), 2020, p. 59–61.

[168] R. Stoyanov, N. Zilberman, MTPSA: Multi-Tenant Programmable Switches, in: P4 Workshop in Europe (EuroP4), 2020, p. 43–48.

[169] GitHub: MTPSA, `https://github.com/mtpsa`, accessed 01-20-2021 (2021).

[170] S. Han, S. Jang, H. Choi, H. Lee, S. Pack, Virtualization in Programmable Data Plane: A Survey and Open Challenges, IEEE Open Journal of the Communications Society 1 (2020) 527–534.

[171] J. Santiago da Silva, T. Stimpfling, T. Luinaud, B. Fradj, B. Boughzala, One for All, All for One: A Heterogeneous Data Plane for Flexible P4 Processing, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 440–441.

[172] C. Beckmann, R. Krishnamoorthy, H. Wang, A. Lam, C. Kim, Hurdles for a DRAM-based Match-Action Table, in: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 13–16.

[173] A. Aghdai, Y. Xu, H. J. Chao, Design of a hybrid modular switch, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2017, pp. 1–6.

[174] S. Laki, D. Horpacsi, P. Voros, M. Tejfel, P. Hudoba, G. Pongracz, L. Molnar, The Price for Asynchronous Execution of Extern Functions in Programmable Software Data Planes, in: Workshop on Flexible Network Data Plane Processing (NETPROC@ICIN), 2020, pp. 23–28.

[175] D. Horpácsi, P. Vörös, M. Tejfel, S. Laki, G. Pongrácz, L. Molnár, Asynchronous Extern Functions in Programmable Software Data Planes, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–2.

[176] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, G. Carle, Cryptographic Hashing in P4 Data Planes, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[177] J. S. da Silva, F.-R. Boyer, L.-O. Chiquette, J. P. Langlois, Extern Objects in P4: an ROHC Header Compression Scheme Case Study, in: IEEE Conference on Network Softwarization (NetSoft), 2018, pp. 517–522.

[178] N. Gray, A. Grigorjew, T. Hosssfeld, A. Shukla, T. Zinner, Highlighting the Gap Between Expected and Actual Behavior in P4-enabled Networks, in: IFIP/IEEE Symposium on Integrated Management (IM), 2019, pp. 731–732.

[179] M. V. Dumitru, D. Dumitrescu, C. Raiciu, Can We Exploit Buggy P4 Programs?, in: ACM Symposium on SDN Research (SOSR), 2020, p. 62–68.

[180] J. Mambretti, J. Chen, F. Yeh, S. Y. Yu, International P4 Networking Testbed, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[181] B. Chung, C. Tseng, J. H. Chen, J. Mambretti, P4MT: Multi-Tenant Support Prototype for International P4 Testbed, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[182] A national programmable infrastructure to experiment with next-generation networks, https://www.2stic.nl/national-programmable-infrastructure.html, accessed 01-20-2021 (2021).

[183] R. Sukapuram, G. Barua, PPCU: Proportional Per-packet Consistent Updates for SDNs using Data Plane Time Stamps, Computer Networks 155 (2019) 72–86.

[184] R. Sukapuram, G. Barua, ProFlow: Proportional Per-Bidirectional-Flow Consistent Updates, IEEE Transactions on Network and Service Management (TNSM) 16 (2019) 675–689.

[185] S. Liu, T. A. Benson, M. K. Reiter, Efficient and Safe Network Updates with Suffix Causal Consistency, in: European Conference on Computer Systems (EUROSYS), 2019, p. 1–15.

[186] T. D. Nguyen, M. Chiesa, M. Canini, Decentralized Consistent Network Updates in SDN with ez-Segway, ArXiv e-prints (2017).

[187] S. Geissler, S. Herrnleben, R. Bauer, A. Grigorjew, T. Zinner, M. Jarschel, The Power of Composition: Abstracting aMulti-Device SDN Data Path Through a Single API, IEEE Transactions on Network and Service Management (TNSM) (2019) 722–735.

[188] E. C. Molero, S. Vissicchio, L. Vanbever, Hardware-Accelerated Network Control Planes, in: ACM Workshop on Hot Topics in Networks (HotNets), 2018, p. 120–126.

[189] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, J. Rexford, Heavy-Hitter Detection Entirely in the Data Plane, in: ACM Symposium on SDN Research (SOSR), 2017, p. 164–176.

[190] GitHub: Hashpipe, https://github.com/vibhaa/hashpipe, accessed 01-20-2021 (2021).

[191] Y. Lin, C. Huang, S. Tsai, SDN Soft Computing Application for Detecting Heavy Hitters, IEEE Transactions on Industrial Informatics (ToII) 15 (2019) 5690–5699.

[192] D. A. Popescu, G. Antichi, A. W. Moore, Enabling Fast Hierarchical Heavy Hitter Detection using Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2017, p. 191–192.

[193] R. Harrison, Q. Cai, A. Gupta, J. Rexford, Network-Wide Heavy Hitter Detection with Commodity Switches, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–7.

[194] J. Kučera, D. A. Popescu, H. Wang, A. Moore, J. Kořenek, G. Antichi, Enabling Event-Triggered Data Plane Monitoring, in: ACM Symposium on SDN Research (SOSR), 2020, p. 14–26.

[195] M. Silva, A. Jacobs, R. Pfitscher, L. Granville, IDEAFIX: Identifying Elephant Flows in P4-Based IXP Networks, in: IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6.

[196] B. Turkovic, J. Oostenbrink, F. Kuipers, Detecting Heavy Hitters in the Data-plane, ArXiv e-prints (2019).

[197] D. Ding, M. Savi, G. Antichi, D. Siracusa, An Incrementally-Deployable P4-Enabled Architecture for Network-Wide Heavy-Hitter Detection, IEEE Transactions on Network and Service Management (TNSM) 17 (2020) 75–88.

[198] GitHub: Network-Wide Heavy-Hitter Detection Implementation in P4 Language, `https://github.com/DINGDAMU/Network-wide-heavy-hitter-detection`, accessed 01-20-2021 (2021).

[199] J. Sonchack, A. J. Aviv, E. Keller, J. M. Smith, Turboflow: Information Rich Flow Record Generation on Commodity Switches, in: European Conference on Computer Systems (EUROSYS), 2018, p. 1–16.

[200] GitHub: TurboFlow, `https://github.com/jsonch/TurboFlow`, accessed 01-20-2021 (2021).

[201] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, J. M. Smith, Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow, in: USENIX Annual Technical Conference (ATC), 2018, pp. 823–835.

[202] GitHub: StarFlow, `https://github.com/jsonch/starflow`, accessed 01-25-2021 (2021).

[203] J. Hill, M. Aloserij, P. Grosso, Tracking Network Flows with P4, in: IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), 2018, pp. 23–32.

[204] L. Castanheira, R. Parizotto, A. E. Schaeffer-Filho, FlowStalker: Comprehensive Traffic Flow Monitoring on the Data Plane using P4, in: IEEE International Conference on Communicaotions (ICC), 2019, pp. 1–6.

[205] R. Parizotto, L. Castanheira, R. H. Ribeiro, L. Zembruzki, A. S. Jacobs, L. Z. Granville, A. Schaeffer-Filho, ShadowFS: Speeding-up Data Plane Monitoring and Telemetry using P4, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[206] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, A. Madeira, FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications, in: Network and Distributed Systems Security Symposium (NDSS), 2021, pp. 1–18.

[207] GitHub: FlowLens, `https://github.com/dmbb/FlowLens`, accessed 04-14-2021 (2021).

[208] W. Wang, P. Tammana, A. Chen, T. S. E. Ng, Grasp the Root Causes in the Data Plane: Diagnosing Latency Problems with SpiderMon, in: ACM Symposium on SDN Research (SOSR), 2020, p. 55–61.

[209] X. Chen, S. Landau-Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, T.-Y. Wang, Fine-Grained Queue Measurement in the Data Plane, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 15–29.

[210] Z. Zhao, X. Shi, X. Yin, Z. Wang, Q. Li, HashFlow for Better Flow Record Collection, in: IEEE International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1416–1425.

[211] Q. Huang, P. P. C. Lee, Y. Bao, Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference, in: ACM SIGCOMM Conference, 2018, p. 576–590.

[212] GitHub: SketchLearn, `https://github.com/huangqundl/SketchLearn`, accessed 01-20-2021 (2021).

[213] L. Tang, Q. Huang, P. C. Lee, A Fast and Compact Invertible Sketch for Network-Wide Heavy Flow Detection, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 2350–2363.

[214] GitHub: MV-Sketch, `https://github.com/Grace-TL/MV-Sketch`, accessed 01-20-2021 (2021).

[215] Z. Hang, M. Wen, Y. Shi, C. Zhang, Interleaved Sketch: Toward Consistent Network Telemetry for Commodity Programmable Switches, IEEE ACCESS 7 (2019) 146745–146758.

[216] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman, One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon, in: ACM SIGCOMM Conference, 2016, p. 101–114.

[217] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Elastic Sketch: Adaptive and Fast Network-wide Measurements, in: ACM SIGCOMM Conference, 2018, p. 561–575.

[218] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Adaptive Measurements Using One Elastic Sketch, IEEE/ACM Transactions on Networking (ToN) 27 (2019) 2236–2251.

[219] GitHub: ElasticSketch, https://github.com/BlockLiu/ElasticSketchCode, accessed 01-20-2021 (2021).

[220] F. Pereira, N. Neves, F. M. V. Ramos, Secure network monitoring using programmable data planes, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2017, pp. 286–291.

[221] R. F. T. Martins, F. L. Verdi, R. Villaça, L. F. U. Garcia, Using Probabilistic Data Structures for Monitoring of Multi-tenant P4-based Networks, in: IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 204–207.

[222] Y.-K. Lai, K.-Y. Shih, P.-Y. Huang, H.-P. Lee, Y.-J. Lin, T.-L. Liu, J. H. Chen, Sketch-based Entropy Estimation for Network Traffic Analysis using Programmable Data Plane ASICs, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[223] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, J. Rexford, Memory-Efficient Performance Monitoring on Programmable Switches with Lean Algorithms, in: SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS), 2020, pp. 31–44.

[224] L. Tang, Q. Huang, P. P. C. Lee, SpreadSketch: Toward Invertible and Network-Wide Detection of Superspreaders, in: IEEE International Conference on Computer Communications (INFOCOM), 2020, pp. 1608–1617.

[225] GitHub: SpreadSketch, http://adslab.cse.cuhk.edu.hk/software/spreadsketch/, accessed 01-20-2021 (2021).

[226] J. Vestin, A. Kassler, D. Bhamare, K. Grinnemo, J. Andersson, G. Pongracz, Programmable Event Detection for In-Band Network Telemetry, in: IEEE International Conference on Cloud Networking (IEEE CloudNet), 2019, pp. 1–6.

[227] S. Wang, Y. Chen, J. Li, H. Hu, J. Tsai, Y. Lin, A Bandwidth-Efficient INT System for Tracking the Rules Matched by the Packets of a Flow, in: IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6.

[228] D. Bhamare, A. Kassler, J. Vestin, M. A. Khoshkholghi, J. Taheri, IntOpt: In-Band Network Telemetry Optimization for NFV Service Chain Monitoring, in: IEEE International Conference on Communicaotions (ICC), 2019, pp. 1–7.

[229] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, Y. Liu, Rapid Detection and Localization of Gray Failures in Data Centers via In-band Network Telemetry, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[230] GitHub: Gray Failures Detection and Localization, https://github.com/graytower/INT_DETECT, accessed 01-20-2021 (2021).

[231] B. Niu, J. Kong, S. Tang, Y. Li, Z. Zhu, Visualize Your IP-Over-Optical Network in Realtime: A P4-Based Flexible Multilayer In-Band Network Telemetry (ML-INT) System, IEEE ACCESS 7 (2019) 82413–82423.

[232] N. S. Kagami, R. I. T. da Costa Filho, L. P. Gaspary, CAPEST: Offloading Network Capacity and Available Bandwidth Estimation to Programmable Data Planes, IEEE Transactions on Network and Service Management (TNSM) 17 (2020) 175–189.

[233] GitHub: Capest, https://github.com/nicolaskagami/capest, accessed 01-20-2021 (2021).

[234] N. Choi, L. Jagadeesan, Y. Jin, N. N. Mohanasamy, M. R. Rahman, K. Sabnani, M. Thottan, Run-time Performance Monitoring, Verification, and Healing of End-to-End Services, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 30–35.

[235] A. Sgambelluri, F. Paolucci, A. Giorgetti, D. Scano, F. Cugini, Exploiting Telemetry in Multi-Layer Networks, in: International Conference on Transparent Optical Networks (ICTON), 2020, pp. 1–4.

[236] Y. Feng, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, N. Duffield, A SmartNIC-Accelerated Monitoring Platform for In-band Network Telemetry, in: IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2020, pp. 1–6.

[237] J. Marques, K. Levchenko, L. Gaspary, IntSight: Diagnosing SLO Violations with in-Band Network Telemetry, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2020, p. 421–434.

[238] GitHub: IntSight, https://github.com/jonadmark/intsight-conext, accessed 01-20-2021 (2021).

[239] D. Suh, S. Jang, S. Han, S. Pack, X. Wang, Flexible sampling-based in-band network telemetry in programmable data plane, ICT Express 6 (2020) 62–65.

[240] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, C. Kim, Language-Directed Hardware Design for Network Performance Monitoring, in: ACM SIGCOMM Conference, 2017, p. 85–98.

[241] V. Nathan, S. Narayana, A. Sivaraman, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, C. Kim, Demonstration of the Marple System for Network Performance Monitoring, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 57–59.

[242] GitHub: Marple, https://github.com/performance-queries/marple, accessed 01-20-2021 (2021).

[243] P. Laffranchini, L. Rodrigues, M. Canini, B. Krishnamurthy, Measurements As First-class Artifacts, in: IEEE International Conference on Computer Communications (INFOCOM), 2019, pp. 415–423.

[244] GitHub: Mafia, https://github.com/paololaff/mafia-sdn, accessed 01-20-2021 (2021).

[245] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, W. Willinger, Sonata: Query-Driven Streaming Network Telemetry, in: ACM Symposium on SDN Research (SOSR), 2018, p. 357–371.

[246] GitHub: SONATA, https://github.com/Sonata-Princeton/SONATA-DEV, accessed 01-20-2021 (2021).

[247] R. Teixeira, R. Harrison, A. Gupta, J. Rexford, PacketScope: Monitoring the Packet Lifecycle Inside a Switch, in: ACM Symposium on SDN Research (SOSR), 2020, p. 76–82.

[248] Y. Gao, Y. Jing, W. Dong, UniROPE: Universal and Robust Packet Trajectory Tracing for Software-Defined Networks, IEEE/ACM Transactions on Networking (ToN) 26 (2018) 2515–2527.

[249] S. Knossen, J. Hill, P. Grosso, Hop Recording and Forwarding State Logging: Two Implementations for Path Tracking in P4, in: IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), 2019, pp. 36–47.

[250] A. Indra Basuki, D. Rosiyadi, I. Setiawan, Preserving Network Privacy on Fine-grain Path-tracking Using P4-based SDN, in: International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), 2020, pp. 129–134.

[251] R. Joshi, T. Qu, M. C. Chan, B. Leong, B. T. Loo, BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks, in: ACM SIGOPS Asia-Pacific Workshop on System (APSys), 2018, pp. 1–8.

[252] GitHub: BurstRadar, https://github.com/harshgondaliya/burstradar, accessed 01-20-2021 (2021).

[253] M. Ghasemi, T. Benson, J. Rexford, Dapper: Data Plane Performance Diagnosis of TCP, in: ACM Symposium on SDN Research (SOSR), 2017, p. 61–74.

[254] C.-H. He, B. Y. Chang, S. Chakraborty, C. Chen, L. C. Wang, A Zero Flow Entry Expiration Timeout P4 Switch, in: ACM Symposium on SDN Research (SOSR), 2018, pp. 1–2.

[255] A. Riesenberg, Y. Kirzon, M. Bunin, E. Galili, G. Navon, T. Mizrahi, Time-Multiplexed Parsing in Marking-Based Network Telemetry, in: ACM International Conference on Systems and Storage (SYSTOR), 2019, p. 80–85.

[256] GitHub: P4 Alternate Marking Algorithm, `https://github.com/AlternateMarkingP4/FlaseClase`, accessed 01-20-2021 (2021).

[257] S. Y. Wang, H. W. Hu, Y. B. Lin, Design and Implementation of TCP-Friendly Meters in P4 Switches, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1885–1898.

[258] R. Kundel, F. Siegmund, J. Blendin, A. Rizk, B. Koldehofe, P4STA: High Performance Packet Timestamping with Programmable Packet Processors, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, p. 1–9.

[259] GitHub: P4STA, `https://github.com/ralfkundel/P4STA`, accessed 01-20-2021 (2021).

[260] R. Hark, D. Bhat, M. Zink, R. Steinmetz, A. Rizk, Preprocessing Monitoring Information on the SDN Data-Plane using P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–6.

[261] D. Ding, M. Savi, D. Siracusa, Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–9.

[262] GitHub: P4Entropy, `https://github.com/DINGDAMU/P4Entropy`, accessed 01-20-2021 (2021).

[263] P. Taffet, J. Mellor-Crummey, Lightweight, Packet-Centric Monitoring of Network Traffic and Congestion Implemented in P4, in: IEEE Symposium on High-Performance Interconnects (HOTI), 2019, pp. 54–58.

[264] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, J. Wu, NetView: Towards On-Demand Network-Wide Telemetry in the Data Center, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[265] J. Bai, M. Zhang, G. Li, C. Liu, M. Xu, H. Hu, FastFE: Accelerating ML-Based Traffic Analysis with Programmable Switches, in: Workshop on Secure Programmable Network Infrastructure (SPIN), 2020, p. 1–7.

[266] J. Kučera, R. B. Basat, M. Kuka, G. Antichi, M. Yu, M. Mitzenmacher, Detecting Routing Loops in the Data Plane, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2020, p. 466–473.

[267] Z. Hang, Y. Shi, M. Wen, C. Zhang, TBSW: Time-Based Sliding Window Algorithm for Network Traffic Measurement, in: IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019, pp. 1305–1310.

[268] B. Guan, S. Shen, FlowSpy: An Efficient Network Monitoring Framework Using P4 in Software-Defined Networks, in: IEEE Semiannual Vehicular Technology Conference (VTC), 2019, pp. 1–5.

[269] Heavy Hitter Detection: Guest lecture for CS344 at Stanford, https://cs344-stanford.github.io/lectures/Lecture-4-HHD.pdf, accessed 01-20-2021 (2018).

[270] B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC 3954, RFC Editor (10 2004).
URL http://www.rfc-editor.org/rfc/rfc3954.txt

[271] P. Phaal, S. Panchen, N. McKee, InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, RFC 3176, RFC Editor (09 2001).
URL http://www.rfc-editor.org/rfc/rfc3176.txt

[272] B. Claise, B. Trammell, P. Aitken, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, STD 77, RFC Editor (09 2013).
URL http://www.rfc-editor.org/rfc/rfc7011.txt

[273] In-band Network Telemetry (INT), https://p4.org/assets/INT-current-spec.pdf, accessed 01-20-2021 (2021).

[274] Charter of the P4 Applications WG, https://github.com/p4lang/p4-applications/blob/master/docs/charter.pdf, accessed 01-20-2021 (2021).

[275] C. Kim, A. Sivaraman, N. P. Katta, A. Bas, A. Dixit, L. J. Wobker, In-band Network Telemetry via Programmable Dataplanes, https://nkatta.github.io/papers/int-demo.pdf (2015).

[276] F. Cugini, P. Gunning, F. Paolucci, P. Castoldi, A. Lord, P4 In-Band Telemetry (INT) for Latency-Aware VNF in Metro Networks, in: Optical Fiber Communication Conference (OFC), 2019, pp. 1–3.

[277] Open Networking Foundation: Trellis, https://www.opennetworking.org/trellis/, accessed 01-20-2021 (2021).

[278] Google Presentations: Trellis & P4 Tutorial, http://bit.ly/trellis-p4-slides, accessed 01-20-2021 (2018).

[279] GitHub: ONF Trellis, https://github.com/opennetworkinglab/routing/tree/master/trellis, accessed 01-20-2021 (2021).

[280] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, M. Budiu, DC.P4: Programming the Forwarding Plane of a Data-center Switch, in: ACM SIGCOMM Conference, 2015, p. 1–8.

[281] GitHub: DC.p4, https://github.com/p4lang/papers/tree/master/sosr15, accessed 01-20-2021 (2021).

[282] Open Network Foundation: P4 apps at ONF, https://github.com/p4lang/p4-applications/blob/master/meeting_slides/2018_04_19_ONF.pdf, accessed 01-20-2021 (2018).

[283] GitHub: fabric.p4, https://github.com/opennetworkinglab/onos/blob/master/pipelines/fabric/impl/src/main/resources/fabric.p4, accessed 01-20-2021 (2021).

[284] RARE (Router for Academia, Research & Education), https://wiki.geant.org/display/RARE/Home, accessed 04-16-2021 (2021).

[285] GitHub: RARE, https://github.com/frederic-loui/RARE, accessed 04-16-2021 (2021).

[286] B. Pit-Claudel, Y. Desmouceaux, P. Pfister, M. Townsley, T. Clausen, Stateless Load-Aware Load Balancing in P4, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 418–423.

[287] R. Miao, H. Zeng, C. Kim, J. Lee, M. Yu, SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap using Switching ASICs, in: ACM SIGCOMM Conference, 2017, p. 15–28.

[288] N. Katta, M. Hira, C. Kim, A. Sivaraman, J. Rexford, HULA: Scalable Load Balancing using Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2016, p. 1–12.

[289] C. H. Benet, A. J. Kassler, T. Benson, G. Pongracz, MP-HULA: Multipath Transport Aware Load Balancing using Programmable Data Planes, in: Morning Workshop on In-Network Computing, 2018, p. 7–13.

[290] B. T. Chiang, K. Wang, Cost-effective Congestion-aware Load Balancing for Datacenters, in: International Conference on Electronics, Information, and Communication (ICEIC), 2019, pp. 1–6.

[291] J.-L. Ye, C. Chen, Y. H. Chu, A Weighted ECMP Load Balancing Scheme for Data Centers using P4 Switches, in: IEEE International Conference on Cloud Networking (IEEE CloudNet), 2018, pp. 1–4.

[292] K.-F. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, D. Walker, Adaptive Weighted Traffic Splitting in Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2020, p. 103–109.

[293] M. Pizzutti, A. Schaeffer-Filho, An Efficient Multipath Mechanism Based on the Flowlet Abstraction and P4, in: IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6.

[294] M. Pizzutti, A. Schaeffer-Filho, Adaptive Multipath Routing based on Hybrid Data and Control Plane Operation, in: IEEE International Conference on Computer Communications (INFOCOM), 2020, pp. 730–738.

[295] J. Zhang, S. Wen, J. Zhang, H. Chai, T. Pan, T. Huang, L. Zhang, Y. Liu, F. R. Yu, Fast Switch-Based Load Balancer Considering Application Server States, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1391–1404.

[296] Q. Li, J. Zhang, T. Pan, T. Huang, Y. Liu, Data-driven Routing Optimization based on Programmable Data Plane, in: IEEE International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1–9.

[297] E. Kawaguchi, H. Kasuga, N. Shinomiya, Unsplittable flow Edge Load factor Balancing in SDN using P4 Runtime, in: International Telecommunication Networks and Applications Conference (ITNAC), 2019, pp. 1–6.

[298] E. Cidon, S. Choi, S. Katti, N. McKeown, AppSwitch: Application-layer Load Balancing withina Software Switch, in: Asia-Pacific Workshop on Networking (APnet), 2017, p. 64–70.

[299] V. Olteanu, A. Agache, A. Voinescu, C. Raiciu, Stateless Datacenter Load-balancing with Beamer, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2018, pp. 125–139.

[300] GitHub: Beamer, `https://github.com/Beamer-LB`, accessed 01-25-2021 (2021).

[301] J. Geng, J. Yan, Y. Zhang, P4QCN: Congestion Control using P4-Capable Device in Data Center Networks, Electronics Journal 8 (2019) 280.

[302] J. Jiang, Y. Zhang, An Accurate Congestion Control Mechanism in Programmable Network, in: IEEE Annual Computing and Communication Workshop and Conference (CCWC), 2019, pp. 673–677.

[303] S. Shahzad, E. Jung, J. Chung, R. Kettimuthu, Enhanced Explicit Congestion Notification (EECN) in TCP with P4 Programming, in: International Conference on Green and Human Information Technology (ICGHIT), 2020, pp. 35–40.

[304] C. Chen, H. Fang, M. S. Iqbal, QoSTCP: Provide Consistent Rate Guarantees to TCP flows in Software Defined Networks, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[305] A. Laraba, J. François, I. Chrisment, S. R. Chowdhury, R. Boutaba, Defeating Protocol Abuse with P4: Application to Explicit Congestion Notification, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 431–439.

[306] N. K. Sharma, M. Liu, K. Atreya, A. Krishnamurthy, Approximating Fair Queueing on Reconfigurable Switches, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2018, p. 1–16.

[307] C. Cascone, N. Bonelli, L. Bianchi, A. Capone, B. Sansò, Towards Approximate Fair Bandwidth Sharing via Dynamic Priority Queuing, in: IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2017, pp. 1–6.

[308] D. Bhat, J. Anderson, P. Ruth, M. Zink, K. Keahey, Application-based QoE support with P4 and OpenFlow, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 817–823.

[309] E. F. Kfoury, J. Crichigno, E. Bou-Harb, D. Khoury, G. Srivastava, Enabling TCP Pacing using Programmable Data Plane Switches, in: International Conference on Telecommunications and Signal Processing (TSP), 2019, pp. 273–277.

[310] Y. Chen, L. Yen, W. Wang, C. Chuang, Y. Liu, C. Tseng, P4-Enabled Bandwidth Management, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2019, pp. 1–5.

[311] S. S. W. Lee, K. Chan, A Traffic Meter Based on a Multicolor Marker for Bandwidth Guarantee and Priority Differentiation in SDN Virtual Networks, IEEE Transactions on Network and Service Management (TNSM) 16 (2019) 1046–1058.

[312] S.-Y. Wang, J.-Y. Li, Y.-B. Lin, Aggregating and disaggregating packets with various sizes of payload in P4 switches at 100 Gbps line rate, Journal of Network and Computer Applications (JNCA) 165 (2020) 102676.

[313] K. Tokmakov, M. Sarker, J. Domaschka, S. Wesner, A Case for Data Centre Traffic Management on Software Programmable Ethernet Switches, in: IEEE International Conference on Cloud Networking (IEEE CloudNet), 2019, pp. 1–6.

[314] B. Turkovic, F. Kuipers, N. van Adrichem, K. Langendoen, Fast Network Congestion Detection and Avoidance using P4, in: Workshop on Networking for Emerging Applications and Technologies (NEAT), 2018, p. 45–51.

[315] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, R. Steinmetz, P4-CoDel: Active Queue Management in Programmable Data Planes, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2018, pp. 1–4.

[316] GitHub: P4-CoDel, `https://github.com/ralfkundel/p4-codel`, accessed 01-20-2021 (2021).

[317] M. Menth, H. Mostafaei, D. Merling, M. Häberle, Implementation and Evaluation of Activity-Based Congestion Management using P4 (P4-ABC), MDPI Future Internet Journal (FI) 11 (2019) 159.

[318] B. Turkovic, F. Kuipers, P4air: Increasing Fairness among Competing Congestion Control Algorithms, in: IEEE International Conference on Network Protocols (ICNP), 2020, pp. 1–12.

[319] L. B. Fernandes, L. Camargos, Bandwidth throttling in a P4 switch, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 91–94.

[320] G. Wang, C. Chen, C. Chen, L. Pan, Y. Wang, C. Fan, C. Hsu, Streaming Scalable Video Sequences with Media-Aware Network Elements Implemented in P4 Programming Language, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–2.

[321] A. G. Alcoz, A. Dietmüller, L. Vanbever, SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 59–76.

[322] I. Kunze, M. Gunz, D. Saam, K. Wehrle, J. Rüth, Tofino + P4: A Strong Compound for AQM on High-Speed Networks?, in: IFIP/IEEE International Symposium on Integrated Network Management, 2021, pp. 72–80.

[323] GitHub: PIE for Tofino, `https://github.com/COMSYS/pie-for-tofino`, accessed 04-15-2021 (2021).

[324] H. Harkous, C. Papagianni, K. De Schepper, M. Jarschel, M. Dimolianis, R. Preis, Virtual queues for p4: A poor man's programmable traffic manager, IEEE Transactions on Network and Service Management (TNSM) (2021) 1–1.

[325] B. Andrus, S. A. Sasu, T. Szyrkowiec, A. Autenrieth, M. Chamania, J. K. Fischer, S. Rasp, Zero-Touch Provisioning of Distributed Video Analytics in a Software-Defined Metro-Haul Network with P4 Processing, in: Optical Fiber Communication Conference (OFC), 2019, pp. 1–3.

[326] S. Ibanez, G. Antichi, G. Brebner, N. McKeown, Event-Driven Packet Processing, in: ACM Workshop on Hot Topics in Networks (HotNets), 2019, p. 133–140.

[327] E. F. Kfoury, J. Crichigno, E. Bou-Harb, Offloading Media Traffic to Programmable Data Plane Switches, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–7.

[328] I. Kettaneh, S. Udayashankar, A. Abdel-hadi, R. Grosman, S. Al-Kiswany, Falcon: Low Latency, Network-Accelerated Scheduling, in: P4 Workshop in Europe (EuroP4), 2020, p. 7–12.

[329] T. Osiński, M. Kossakowski, M. Pawlik, J. Palimąka, M. Sala, H. Tarasiuk, Unleashing the Performance of Virtual BNG by Offloading Data Plane to a Programmable ASIC, in: P4 Workshop in Europe (EuroP4), 2020, p. 54–55.

[330] J. Lee, R. Miao, C. Kim, M. Yu, H. Zeng, Stateful Layer-4 Load Balancing in Switching ASICs, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 133–135.

[331] K. Nichols, V. Jacobson, A. McGregor, J. Iyengar, Controlled Delay Active Queue Management, RFC 8289, RFC Editor (01 2018).
URL `https://tools.ietf.org/rfc/rfc8289.txt`

[332] B. Lewis, L. Fawcett, M. Broadbent, N. Race, Using P4 to Enable Scalable Intents in Software Defined Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 442–443.

[333] GitHub: P4 Source Routing, `https://github.com/BenRLewis/P4-Source-Routing`, accessed 01-20-2021 (2021).

[334] L. Luo, H. Yu, S. Luo, Z. Ye, X. Du, M. Guizani, Scalable Explicit Path Control in Software-Defined Networks, Journal of Network and Computer Applications (JNCA) 141 (2019) 86–103.

[335] GitHub: P4 Paco, `https://github.com/an15m/paco`, accessed 01-20-2021 (2021).

[336] A. Kushwaha, S. Sharma, N. Bazard, A. Gumaste, B. Mukherjee, Design, Analysis, and a Terabit Implementation of a Source-Routing-Based SDN Data Plane, IEEE Systems Journal (2020).

[337] A. Abdelsalam, A. Tulumello, M. Bonola, S. Salsano, C. Filsfils, Pushing Network Programmability to the limits with SRv6 uSIDs and P4, in: P4 Workshop in Europe (EuroP4), 2020, p. 62–64.

[338] W. Braun, J. Hartmann, M. Menth, Demo: Scalable and Reliable Software-Defined Multicast with BIER and P4, in: IFIP/IEEE Symposium on Integrated Management (IM), 2017, pp. 905–906.

[339] Bitbucket: p4-bfr), `https://bitbucket.org/wb-ut/p4-bfr`, accessed 01-20-2021 (2021).

[340] D. Merling, S. Lindner, M. Menth, P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast, Journal of Network and Computer Applications (JNCA) 169 (2020) 102764.

[341] D. Merling, S. Lindner, M. Menth, Hardware-based evaluation of scalable and resilient multicast with bier in p4, IEEE ACCESS 9 (2021) 34500–34514.

[342] GitHub: P4-BIER, `https://github.com/uni-tue-kn/p4-bier`, accessed 01-20-2021 (2021).

[343] GitHub: P4-BIER for Tofino, `https://github.com/uni-tue-kn/p4-bier-tofino`, accessed 04-26-2021 (2021).

[344] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, M. Hira, Elmo: Source Routed Multicast for Public Clouds, in: ACM Special Interest Group on Data Communication, 2019, pp. 2587–2600.

[345] GitHub: Elmo MCast, `https://github.com/Elmo-MCast/p4-programs`, accessed 01-20-2021 (2021).

[346] S. Luo, H. Yu, K. Li, H. Xing, Efficient File Dissemination in Data Center Networks with Priority-based Adaptive Multicast, IEEE Journal on Selected Areas in Communications (JSAC) 38 (2020) 1161–1175.

[347] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, D. Timmermann, Realizing Content-Based Publish/Subscribe with P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2018, pp. 1–7.

[348] C. Wernecke, H. Parzyjegla, G. Mühl, E. Schweissguth, D. Timmermann, Flexible Notification Forwarding for Content-Based Publish/Subscribe Using P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 1–5.

[349] C. Wernecke, H. Parzyjegla, G. Mühl, Implementing Content-based Publish/Subscribe on the Network Layer with P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 144–149.

[350] C. Wernecke, H. Parzyjegla, G. Mühl, P. Danielis, E. Schweissguth, D. Timmermann, Stitching Notification Distribution Trees for Content-based Publish/Subscribe with P4, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2020, pp. 100–104.

[351] T. Jepsen, M. Moshref, A. Carzaniga, N. Foster, R. Soulé, Packet Subscriptions for Programmable ASICs, in: ACM Workshop on Hot Topics in Networks (HotNets), 2018, p. 176–183.

[352] R. Kundel, C. Gaertner, M. Luthra, S. Bhowmik, B. Koldehofe, Flexible Content-based Publish/Subscribe over Programmable Data Planes, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2020, pp. 1–5.

[353] GitHub: p4bsub, `https://github.com/ralfkundel/p4bsub/`, accessed 01-20-2021 (2021).

[354] J. Vestin, A. Kassler, S. Laki, G. Pongrácz, Towards In-Network Event Detection and Filtering for Publish/Subscribe Communication using Programmable Data Planes, IEEE Transactions on Network and Service Management (TNSM) (2020) 415–428.

[355] S. Signorello, R. State, J. François, O. Festor, NDN.p4: Programming Information-Centric Data-Planes, in: IEEE Conference on Network Softwarization (NetSoft), 2016, pp. 384–389.

[356] R. Miguel, S. Signorello, F. M. V. Ramos, Named Data Networking with Programmable Switches, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 400–405.

[357] GitHub: NDN.p4, `https://github.com/signorello/NDN.p4`, accessed 01-20-2021 (2021).

[358] GitHub: NDN.p4-16, `https://github.com/netx-ulx/NDN.p4-16`, accessed 01-20-2021 (2021).

[359] O. Karrakchou, N. Samaan, A. Karmouch, ENDN: An Enhanced NDN Architecture with a P4-programmable Data Plane, in: International Conference on Networking (ICN), 2020, p. 1–11.

[360] R. Sedar, M. Borokhovich, M. Chiesa, G. Antichi, S. Schmid, Supporting Emerging Applications With Low-Latency Failover in P4, in: Workshop on Networking for Emerging Applications and Technologies (NEAT), 2018, p. 52–57.

[361] GitHub: P4-FRR, `https://bitbucket.org/roshanms/p4-frr/src/master/`, accessed 01-20-2021 (2021).

[362] H. Giesen, L. Shi, J. Sonchack, A. Chelluri, N. Prabhu, N. Sultana, L. Kant, A. J. McAuley, A. Poylisher, A. DeHon, B. T. Loo, In-Network Computing to the Rescue of Faulty Links, in: Morning Workshop on In-Network Computing, 2018, pp. 1–6.

[363] T. Qu, R. Joshi, M. Chan, B. Leong, D. Guo, Z. Liu, SQR: In-network Packet Loss Recovery from Link Failures for Highly Reliable Datacenter Networks, in: IEEE International Conference on Network Protocols (ICNP), 2019, pp. 1–12.

[364] GitHub: P4 SQR, `https://git.io/fjbnV`, accessed 01-20-2021 (2021).

[365] S. Lindner, D. Merling, M. Häberle, M. Menth, P4-Protect: 1+1 Path Protection for P4, in: P4 Workshop in Europe (EuroP4), 2020, p. 21–27.

[366] GitHub: P4-Protect BMv2, https://github.com/uni-tue-kn/p4-protect, accessed 01-20-2021 (2021).

[367] GitHub: P4-Protect Tofino, https://github.com/uni-tue-kn/p4-protect-tofino, accessed 01-20-2021 (2021).

[368] K. Hirata, , T. Tachibana, Implementation of Multiple Routing Configurations on Software-Defined Networks with P4, in: Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2019, pp. 13–16.

[369] S. Lindner, M. Häberle, F. Heimgaertner, N. Nayak, S. Schildt, D. Grewe, H.Loehr, M. Ment, P4 In-Network Source Protection for Sensor Failover, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 791–796.

[370] GitHub: P4 Source Protection BMv2, https://github.com/uni-tue-kn/p4-source-protection, accessed 01-20-2021 (2021).

[371] GitHub: P4 Source Protection Tofino, https://github.com/uni-tue-kn/p4-source-protection-tofino, accessed 01-20-2021 (2021).

[372] K. Subramanian, A. Abhashkumar, L. D'Antoni, A. Akella, D2R: Dataplane-Only Policy-Compliant Routing Under Failures (2019).

[373] M. Chiesa, R. Sedar, G. Antichi, M. Borokhovich, A. Kamisiński, G. Nikolaidis, S. Schmid, PURR: A Primitive for Reconfigurable Fast Reroute, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 1–14.

[374] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, L. Vanbever, Blink: Fast Connectivity Recovery Entirely in the Data Plane, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2019, pp. 161–176.

[375] GitHub: Blink, https://github.com/nsg-ethz/Blink, accessed 01-20-2021 (2021).

[376] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, D. Walker, Contra: A Programmable System for Performance-aware Routing, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 701–721.

[377] O. Michel, E. Keller, Policy Routing using Process-Level Identifiers, in: IEEE International Conference on Cloud Engineering Workshop (IC2EW), 2016, pp. 7–12.

[378] A. C. Baktir, A. Ozgovde, C. Ersoy, Implementing Service-Centric Model with P4: A Fully-Programmable Approach, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–6.

[379] W. Froes, L. Santos, L. N. Sampaio, M. Martinello, A. Liberato, R. S. Villaca, ProgLab: Programmable Labels for QoS Provisioning on Software Defined Networks, Computer Communications 161 (2020) 99–108.

[380] N. VARYANI, Z.-L. ZHANG, D. DAI, QROUTE: An Efficient Quality of Service (QoS) Routing Scheme for Software-Defined Overlay Networks, IEEE ACCESS 8 (2020) 104109–104126.

[381] S. Gimenez, E. Grasa, S. Bunch, A Proof of Concept Implementation of a RINA Interior Router using P4-enabled Software Targets, in: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 57–62.

[382] W. Feng, X. Tan, Y. Jin, Implementing ICN over P4 in HTTP Scenario, in: IEEE International Conference on Hot Information-Centric Networking (HotICN), 2019, pp. 37–43.

[383] G. Grigoryan, Y. Liu, M. Kwon, PFCA: A Programmable FIB Caching Architecture, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1872–1884.

[384] A. McAuley, Y. M. Gottlieb, L. Kant, J. Lee, A. Poylisher, P4-Based Hybrid Error Control Booster Providing New Design Tradeoffs in Wireless Networks, in: IEEE Military Communications Conference (MILCOM), 2019, pp. 731–736.

[385] M. Kogias, G. Prekas, A. Ghosn, J. Fietz, E. Bugnion, R2P2: Making RPCs first-class datacenter citizens, in: USENIX Annual Technical Conference (ATC), 2019, pp. 863–880.

[386] GitHub: R2P2 - Request Response Pair Protocol, `https://github.com/epfl-dcsl/r2p2`, accessed 01-25-2021 (2021).

[387] D. Merling, M. Menth, N. Warnke, T. Eckert, An Overview of Bit Index Explicit Replication (BIER), IETF Journal (2018).

[388] M. Hollingsworth, J. Lee, Z. Liu, J. Lee, S. Ha, D. Grunwald, P4EC: Enabling Terabit Edge Computing in Enterprise 4G LTE, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge), 2020, pp. 1–7.

[389] GitHub: spgw.p4, `https://github.com/opennetworkinglab/onos/blob/master/pipelines/fabric/impl/src/main/resources/include/control/spgw.p4`, accessed 01-20-2021 (2021).

[390] P. Palagummi, K. M. Sivalingam, SMARTHO: A Network Initiated Handover in NG-RAN using P4-based Switches, in: International Conference on Network and Services Management (CNSM), 2018, pp. 338–342.

[391] A. Aghdai, M. Huang, D. Dai, Y. Xu, J. Chao, Transparent Edge Gateway for Mobile Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 412–417.

[392] A. Aghdai, Y. Xu, M. Huang, D. H. Dai, H. J. Chao, Enabling Mobility in LTE-Compatible Mobile-edge Computing with Programmable Switches, ArXiv e-prints (2019).

[393] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, H. Chen, Efficient Data Placement and Retrieval Services in Edge Computing, in: IEEE International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1029–1039.

[394] J. Xie, D. Guo, X. Shi, H. Cai, C. Qian, H. Chen, A Fast Hybrid Data Sharing Framework for Hierarchical Mobile Edge Computing, in: IEEE International Conference on Computer Communications (INFOCOM), 2020, pp. 2609–2618.

[395] C. Shen, D. Lee, C. Ku, M. Lin, K. Lu, S. Tan, A Programmable and FPGA-accelerated GTP Offloading Engine for Mobile Edge Computing in 5G Networks, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2019, pp. 1021–1022.

[396] C. Lee, K. Ebisawa, H. Kuwata, M. Kohno, S. Matsushima, Performance Evaluation of GTP-U and SRv6 Stateless Translation, in: International Conference on Network and Services Management (CNSM), 2019, pp. 1–6.

[397] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, P4-NetFPGA-based network slicing solution for 5G MEC architectures, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–2.

[398] S. K. Singh, C. E. Rothenberg, G. Patra, G. Pongracz, Offloading Virtual Evolved Packet Gateway User Plane Functions to a Programmable ASIC, in: ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, 2019, p. 9–14.

[399] R. Shah, V. Kumar, M. Vutukuru, P. Kulkarni, TurboEPC: Leveraging Dataplane Programmability to Accelerate the Mobile Packet Core, in: ACM Symposium on SDN Research (SOSR), 2020, p. 83–95.

[400] P. Vörös, G. Pongrácz, S. Laki, Towards a Hybrid Next Generation NodeB, in: P4 Workshop in Europe (EuroP4), 2020, p. 56–58.

[401] Y. Lin, T. Huang, S. Tsai, Enhancing 5G/IoT Transport Security Through Content Permutation, IEEE ACCESS 7 (2019) 94293–94299.

[402] M. Uddin, S. Mukherjee, H. Chang, T. V. Lakshman, SDN-Based Service Automation for IoT, in: IEEE International Conference on Network Protocols (ICNP), 2017, pp. 1–10.

[403] M. Uddin, S. Mukherjee, H. Chang, T. V. Lakshman, SDN-Based Multi-Protocol Edge Switching for IoT Service Automation, IEEE Journal on Selected Areas in Communications (JSAC) 36 (2018) 2775–2786.

[404] S.-Y. Wang, C.-M. Wu, Y.-B. Linm, C.-C. Huang, High-Speed Data-Plane Packet Aggregation and Disaggregation by P4 Switches, Journal of Network and Computer Applications (JNCA) 142 (2019) 98–110.

[405] A. L. R. Madureira, F. R. C. Araújo, L. N. Sampaio, On supporting IoT data aggregation through programmable data planes, Computer Networks 177 (2020) 107330.

[406] P. Engelhard, A. Zachlod, J. Schulz-Zander, S. Du, Toward scalable and virtualized massive wireless sensor networks, in: International Conference on Networked Systems (NetSys), 2019, pp. 1–6.

[407] J. Vestin, A. Kassler, J. Åkerberg, FastReact: In-Network Control and Caching for Industrial Control Networks using Programmable Data Planes, in: IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2018, pp. 219–226.

[408] F. E. R. Cesen, L. Csikor, C. Recalde, C. E. Rothenberg, G. Pongrácz, Towards Low Latency Industrial Robot Control in Programmable Data Planes, in: IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 165–169.

[409] I. Kunze, R. Glebke, J. Scheiper, M. Bodenbenner, R. H. Schmitt, K. Wehrle, Investigating the Applicability of In-Network Computing to Industrial Scenarios, in: International Conference on Industrial Cyber-Physical Systems (ICPS), 2021, pp. 334–340.

[410] J. Rüth, R. Glebke, K. Wehrle, V. Causevic, S. Hirche, Towards In-Network Industrial Feedback Control, in: Morning Workshop on In-Network Computing, 2018, p. 14–19.

[411] P. G. Kannan, R. Joshi, M. C. Chan, Precise Time-Synchronization in the Data-Plane using Programmable Switching ASICs, in: ACM Symposium on SDN Research (SOSR), 2019, p. 8–20.

[412] R. Kundel, F. Siegmund, B. Koldehofe, How to Measure the Speed of Light with Programmable Data Plane Hardware?, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–2.

[413] G. Bonofiglio, V. Iovinella, G. Lospoto, G. D. Battista, Kathará: A Container-Based Framework for Implementing Network Function Virtualization and Software Defined Networks, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–9.

[414] M. He, A. Basta, A. Blenk, N. Deric, W. Kellerer, P4NFV: An NFV Architecture with Flexible Data Plane Reconfiguration, in: International Conference on Network and Services Management (CNSM), 2018, pp. 90–98.

[415] T. Osiński, H. Tarasiuk, M. Kossakowski, R. Picard, Offloading Data Plane Functions to the Multi-Tenant Cloud Infrastructure using P4, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[416] D. Moro, G. Verticale, A. Capone, A Framework for Network Function Decomposition and Deployment, in: International Workshop on the Design of Reliable Communication Networks (DRCN), 2020, pp. 1–6.

[417] T. Osiński, H. Tarasiuk, L. Rajewski, E. Kowalczyk, DPPx: A P4-based Data Plane Programmability and Exposure framework to enhance NFV services, in: IEEE Conference on Network Softwarization (NetSoft), 2019, pp. 296–300.

[418] A. Mohammadkhan, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, L. N. Bhuyan, P4NFV: P4 Enabled NFV Systems with SmartNICs, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–7.

[419] D. Moro, M. Peuster, H. Karl, A. Capone, FOP4: Function Offloading Prototyping in Heterogeneous and Programmable Network Scenarios, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–6.

[420] D. Moro, M. Peuster, H. Karl, A. Capone, Demonstrating FOP4: A Flexible Platform to Prototype NFV Offloading Scenarios, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–2.

[421] D. R. Mafioletti, C. K. Dominicini, M. Martinello, M. R. N. Ribeiro, R. d. S. Villaça, Piaffe: A place-as-you-go in-network framework for flexible embedding of vnfs, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[422] X. Chen, D. Zhang, X. Wang, K. Zhu, H. Zhou, P4SC: Towards High-Performance Service Function Chain Implementation on the P4-Capable Device, in: IFIP/IEEE Symposium on Integrated Management (IM), 2019, pp. 1–9.

[423] D. Zhang, X. Chen, Q. Huang, X. Hong, C. Wu, H. Zhou, Y. Yang, H. Liu, Y. Chen, P4SC: A High Performance and Flexible Framework for Service Function Chain, IEEE ACCESS 7 (2019) 160982–160997.

[424] GitHub: P4SC, `https://github.com/P4SC/p4sc`, accessed 01-20-2021 (2021).

[425] H. Lee, J. Lee, H. Ko, S. Pack, Resource-Efficient Service Function Chaining in Programmable Data Plane, in: P4 Workshop in Europe (EuroP4), 2019.

[426] Y. Zhou, J. Bi, C. Zhang, M. Xu, J. Wu, FlexMesh: Flexibly Chaining Network Functions on Programmable Data Planes at Runtime, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 73–81.

[427] A. Stockmayer, S. Hinselmann, M. Häberle, M. Menth, Service Function Chaining Based on Segment Routing Using P4 and SR-IOV (P4-SFC), in: Workshop on Virtualization in High-Performance Cloud Computing (VHPC), 2020, pp. 297–309.

[428] GitHub: P4-SFC, `https://github.com/uni-tue-kn/p4-sfc-faas`, accessed 01-20-2021 (2021).

[429] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, Hardware-Accelerated Firewall for 5G Mobile Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 446–447.

[430] Ruben Ricart-Sanchez and Pedro Malagon and Jose M. Alcaraz-Calero and Qi Wang, NetFPGA-Based Firewall Solution for 5G Multi-Tenant Architectures, in: IEEE International Conference on Edge Computing (EDGE), 2019, pp. 132–136.

[431] J. Cao, J. Bi, Y. Zhou, C. Zhang, CoFilter: A High-Performance Switch-Assisted Stateful Packet Filter, in: ACM SIGCOMM Conference Posters and Demos, 2018, p. 9–11.

[432] R. Datta, S. Choi, A. Chowdhary, Y. Park, P4Guard: Designing P4 Based Firewall, in: IEEE Military Communications Conference (MILCOM), 2018, pp. 1–6.

[433] P. Vörös, A. Kiss, Security Middleware Programming Using P4, in: International Conference on Human Aspects of Information Security, Privacy, and Trust (HAS), 2016, pp. 277–287.

[434] E. O. Zaballa, D. Franco, Z. Zhou, M. S. Berger, P4Knocking: Offloading host-based firewall functionalities to the network, in: Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020, pp. 7–12.

[435] A. Almaini, A. Al-Dubai, I. Romdhani, M. Schramm, Delegation of Authentication to the Data Plane in Software-Defined Networks, in: IEEE International Conferences on Smart Computing, Networking and Services (SmartCNS), 2019, pp. 58–65.

[436] G. Grigoryan, Y. Liu, LAMP: Prompt Layer 7 Attack Mitigation with Programmable Data Planes, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2018, pp. 1–4.

[437] A. Febro, H. Xiao, J. Spring, Telephony Denial of Service Defense at Data Plane (TDoSD@DP), in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018, pp. 1–6.

[438] A. Febro, H. Xiao, J. Spring, Distributed SIP DDoS Defense with P4, in: IEEE Wireless Communications and Networking Conference (WCNC), 2019, pp. 1–8.

[439] M. Kuka, K. Vojanec, J. Kučera, P. Benáček, Accelerated DDoS Attacks Mitigation using Programmable Data Plane, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–3.

[440] F. Paolucci, F. Cugini, P. Castoldi, P4-based Multi-Layer Traffic Engineering Encompassing Cyber Security, in: Optical Fiber Communication Conference (OFC), 2018, pp. 1–3.

[441] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, P. Castoldi, An efficient pipeline processing scheme for programming Protocol-independent Packet Processors, IEEE/OSA Journal of Optical Communications and Networking 11 (2019) 88–95.

[442] Y. Mi, A. Wang, ML-Pushback: Machine Learning Based Pushback Defense Against DDoS, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 80–81.

[443] Y. Afek, A. Bremler-Barr, L. Shafir, Network Anti-Spoofing with SDN Data Plane, in: IEEE International Conference on Computer Communications (INFOCOM), 2017, pp. 1–9.

[444] A. C. Lapolli, J. A. Marques, L. P. Gaspary, Offloading Real-time DDoS Attack Detection to Programmable Data Planes, in: IFIP/IEEE Symposium on Integrated Management (IM), 2019, pp. 19–27.

[445] GitHub: ddosd-p4, `https://github.com/aclapolli/ddosd-p4`, accessed 01-20-2021 (2021).

[446] Y.-Z. Cai, C.-H. Lai, Y.-T. Wang, M.-H. Tsai, Improving Scanner Data Collection in P4-based SDN, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 126–131.

[447] T.-Y. Lin, J.-P. Wu, P.-H. Hung, C.-H. Shao, Y.-T. Wang, Y.-Z. Cai, M.-H. Tsai, Mitigating SYN flooding Attack and ARP Spoofing in SDN Data Plane, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 114–119.

[448] F. Musumeci, V. Ionata, F. Paolucci, M. Cugini, Filippo Tornatore, Machine-learning-assisted DDoS attack detection with P4 language, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–6.

[449] X. Z. Khooi, L. Csikor, D. M. Divakaran, M. S. Kang, DIDA: Distributed In-Network Defense Architecture Against Amplified Reflection DDoS Attacks, in: IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 277–281.

[450] M. Dimolianis, A. Pavlidis, V. Maglaris, A Multi-Feature DDoS Detection Schema on P4 Network Hardware, in: Workshop on Flexible Network Data Plane Processing (NETPROC@ICIN), 2020, pp. 1–6.

[451] D. Scholz, S. Gallenmüller, H. Stubbe, G. Carle, SYN Flood Defense in Programmable Data Planes, in: P4 Workshop in Europe (EuroP4), 2020, p. 13–20.

[452] GitHub: syn-proxy, `https://github.com/syn-proxy`, accessed 01-20-2021 (2021).

[453] K. Friday, E. Kfoury, E. Bou-Harb, J. Crichigno, Towards a Unified In-Network DDoS Detection and Mitigation Strategy, in: IEEE Conference on Network Softwarization (NetSoft), 2020, pp. 218–226.

[454] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, M. Vechev, NetHide: Secure and Practical Network Topology Obfuscation, in: USENIX Security Symposium, 2018, pp. 693–709.

[455] Benjamin Lewis and Matthew Broadbent and Nicholas Race, P4ID: P4 Enhanced Intrusion Detection, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2019, pp. 1–4.

[456] Gorby Kabasele Ndonda and Ramin Sadre, A Two-level Intrusion Detection System for Industrial Control System Networks using P4, in: International Symposium for ICS & SCADA Cyber Security Research (ICS-CSR), 2018, pp. 1–10.

[457] J. Hypolite, J. Sonchack, S. Hershkop, N. Dautenhahn, A. DeHon, J. M. Smith, DeepMatch: Practical Deep Packet Inspection in the Data Plane Using Network Processors, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2020, p. 336–350.

[458] GitHub: DeepMatch, `https://github.com/jhypolite/DeepMatch`, accessed 01-20-2021 (2021).

[459] Q. Qin, K. Poularakis, K. K. Leung, L. Tassiulas, Line-Speed and Scalable Intrusion Detection at the Network Edge via Federated Learning, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 352–360.

[460] GitHub: syn-proxy, `https://github.com/vxxx03/IFIPNetworking20`, accessed 01-20-2021 (2021).

[461] J. Amado, S. Signorello, M. Correia, F. Ramos, Poster: Speeding up network intrusion detection, in: IEEE International Conference on Network Protocols (ICNP), 2020, pp. 1–2.

[462] D. Chang, W. Sun, Y. Yang, A SDN Proactive Defense Mechanism Based on IP Transformation, in: International Conference on Safety Produce Informatization (IICSPI), 2019, pp. 248–251.

[463] W. Feng, Z.-L. Zhang, C. Liu, J. Chen, Clé: Enhancing Security with Programmable Dataplane Enabled Hybrid SDN, in: ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2019, p. 76–77.

[464] P. Kuang, Y. Liu, L. He, P4DAD: Securing Duplicate Address Detection Using P4, in: IEEE International Conference on Communicaotions (ICC), 2020, pp. 1–7.

[465] X. Chen, Implementing aes encryption on programmable switches via scrambled lookup tables, in: Workshop on Secure Programmable Network Infrastructure (SPIN), 2020, p. 8–14.

[466] GitHub: Tofino AES encryption, `https://github.com/Princeton-Cabernet/p4-projects/tree/master/AES-tofino`, accessed 01-20-2021 (2021).

[467] H. Gondaliya, G. C. Sankaran, K. M. Sivalingam, Comparative Evaluation of IP Address Anti-Spoofing Mechanisms Using a P4/NetFPGA-Based Switch, in: P4 Workshop in Europe (EuroP4), 2020, p. 1–6.

[468] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, X. Luo, Programmable In-Network Security for Context-aware BYOD Policies, in: USENIX Security Symposium, 2020, pp. 595–612.

[469] GitHub: Poise, `https://github.com/qiaokang92/poise`, accessed 01-20-2021 (2021).

[470] F. Hauser, M. Schmidt, M. Häberle, M. Menth, P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN, IEEE ACCESS 8 (2020) 58845–58858.

[471] GitHub: P4-MACsec, `https://github.com/uni-tue-kn/p4-macsec`, accessed 01-20-2021 (2021).

[472] F. Hauser, M. Häberle, M. Schmidt, M. Menth, P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN, IEEE ACCESS 8 (2020) 139567–139586.

[473] GitHub: P4-IPsec, `https://github.com/uni-tue-kn/p4-ipsec`, accessed 01-20-2021 (2021).

[474] T. Datta, N. Feamster, J. Rexford, L. Wang, SPINE: Surveillance Protection in the Network Elements, in: USENIX Workshop on Free and Open Communications on the Internet (FOCI), 2019, pp. 1–7.

[475] GitHub: SPINE, `https://github.com/SPINE-P4/spine-code`, accessed 01-20-2021 (2021).

[476] Y. Qin, W. Quan, F. Song, L. Zhang, G. Liu, M. Liu, C. Yu, Flexible Encryption for Reliable Transmission Based on the P4 Programmable Platform, in: Information Communication Technologies Conference (ICTC), 2020, pp. 147–152.

[477] G. Liu, W. Quan, N. Cheng, N. Lu, H. Zhang, X. Shen, P4NIS: Improving network immunity against eavesdropping with programmable data planes, in: IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020, pp. 91–96.

[478] GitHub: P4NIS, `https://github.com/KB00100100/P4NIS`, accessed 01-20-2021 (2021).

[479] M. Liu, D. Gao, G. Liu, J. He, L. Jin, C. Zhou, F. Yang, Learning based adaptive network immune mechanism to defense eavesdropping attacks, IEEE ACCESS 7 (2019) 182814–182826.

[480] J. Deng, H. Hu, H. Li, Z. Pan, K. Wang, G. Ahn, J. Bi, Y. Park, VNGuard: An NFV/SDN Combination Framework for Provisioning and Managing Virtual Firewalls, in: IEEE Conference on Network Function Virtualization and Software-Defined Networking (NFV-SDN), 2015, pp. 107–114.

[481] H. Zhang, W. Quan, H.-c. Chao, C. Qiao, Smart identifier network: A collaborative architecture for the future internet, Networks Magazine 30 (3) (2016) 46–51.

[482] R. Kumar, V. Babu, D. Nicol, Network Coding for Critical Infrastructure Networks, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 436–437.

[483] GitHub: AquaFlow, `https://github.com/gopchandani/AquaFlow`, accessed 01-20-2021 (2021).

[484] D. Goncalves, S. Signorello, F. M. V. Ramos, M. Medard, Random Linear Network Coding on Programmable Switches, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2019, pp. 1–6.

[485] T. Kohler, R. Mayer, F. Dürr, M. Maaß, S. Bhowmik, K. Rothermel, P4CEP: Towards In-Network Complex Event Processing, in: Morning Workshop on In-Network Computing, 2018, p. 33–38.

[486] A. Sapio, I. Abdelaziz, M. Canini, P. Kalnis, DAIET: A System for Data Aggregation Inside the Network, in: ACM Symposium on Cloud Computing (SoCC), 2017, p. 1.

[487] G. C. Sankaran, K. M. Sivalingam, Design and Analysis of Fast IP Address-Lookup Schemes based on Cooperation among Routers, in: International Conference on COMmunication Systems and NETworks (COMSNETS), 2020, pp. 330–339.

[488] Y. Zhang, B. Han, Z.-L. Zhang, V. Gopalakrishnan, Network-Assisted Raft Consensus Algorithm, in: ACM SIGCOMM Conference Posters and Demos, 2017, p. 94–96.

[489] H. T. Dang, M. Canini, F. Pedone, R. Soulé, Paxos Made Switch-y, ACM SIGCOMM Computer Communications Review (CCR) 46 (2016) 18–24.

[490] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilbermanand, H. Weatherspoon, M. Canini, F. Pedone, R. Soulé, P4xos: Consensus as a Network Service, IEEE/ACM Transactions on Networking (ToN) 28 (2020) 1726–1738.

[491] GitHub: P4xos, `https://github.com/P4xos/P4xos`, accessed 01-20-2021 (2021).

[492] E. Sakic, N. Deric, E. Goshi, W. Kellerer, P4BFT: Hardware-Accelerated Byzantine-Resilient Network Control Plane, in: IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–7.

[493] E. Sakic, N. Deric, C. B. Serna, E. Goshi, W. Kellerer, P4BFT: A Demonstration of Hardware-Accelerated BFT in Fault-Tolerant Network Control Plane, in: ACM SIGCOMM Conference Posters and Demos, 2019, p. 6–8.

[494] L. Zeno, D. R. K. Ports, J. Nelson, M. Silberstein, SwiShmem: Distributed Shared State Abstractions for Programmable Switches, in: ACM Workshop on Hot Topics in Networks (HotNets), 2020, p. 160–167.

[495] S. Han, S. Jang, H. Lee, S. Pack, Switch-Centric Byzantine Fault Tolerance Mechanism in Distributed Software Defined Networks, IEEE Communications Letters 24 (2020) 2236–2239.

[496] GitHub: SC-BFT, `https://github.com/MNC-KOR/SC-BFT`, accessed 01-20-2021 (2021).

[497] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, G. Bianchi, LODGE: LOcal Decisions on Global statEs in Progrananaable Data Planes, in: IEEE Conference on Network Softwarization (NetSoft), 2018, pp. 257–261.

[498] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, G. Bianchi, LOcAl DEcisions on Replicated States (LOADER) in programmable dataplanes: Programming abstraction and experimental evaluation, Computer Networks 181 (2020) 107637.

[499] GitHub: LOADER, `https://github.com/german-sv/loader`, accessed 01-20-2021 (2021).

[500] H. Takruri, I. Kettaneh, A. Alquraan, S. Al-Kiswany, FLAIR: Accelerating Reads with Consistency-Aware Network Routing, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2020, pp. 723–737.

[501] S. Luo, H. Yu, L. Vanbever, Swing State: Consistent Updates for Stateful and Programmable Data Planes, in: ACM Symposium on SDN Research (SOSR), 2017, p. 115–121.

[502] J. Xing, A. Chen, T. E. Ng, Secure State Migration in the Data Plane, in: Workshop on Secure Programmable Network Infrastructure (SPIN), 2020, p. 28–34.

[503] GitHub: P4Sync, `https://github.com/jiarong0907/P4Sync`, accessed 01-20-2021 (2021).

[504] Y. Xue, Z. Zhu, Hybrid Flow Table Installation: Optimizing Remote Placements of Flow Tables on Servers to Enhance PDP Switches for In-Network Computing, IEEE Transactions on Network and Service Management (TNSM) (2020) 429–440.

[505] C. Kuzniar, M. Neves, I. Haque, POSTER: Accelerating Encrypted Data Stores Using Programmable Switches, in: IEEE International Conference on Network Protocols (ICNP), 2020, pp. 1–2.

[506] G. C. Sankaran, K. M. Sivalingam, Collaborative Packet Header Parsing in NetFPGA-Based High Speed Switches, IEEE Networking Letters 2 (2020) 124–127.

[507] J. Woodruff, M. Ramanujam, N. Zilberman, P4DNS: In-Network DNS, in: P4 Workshop in Europe (EuroP4), 2019, pp. 1–6.

[508] GitHub: P4DNS, `https://github.com/cucl-srg/P4DNS`, accessed 01-20-2021 (2021).

[509] R. Kundel, L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, V. Gurevich, B. Koldehofe, R. Steinmetz, P4-BNG: Central Office Network Functions on Programmable Packet Pipelines, in: International Conference on Network and Services Management (CNSM), 2019, pp. 1–9.

[510] GitHub: p4se, `https://github.com/opencord/p4se`, accessed 01-20-2021 (2021).

[511] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares, E. Rojas, ARP-P4: A Hybrid ARP-Path/P4Runtime Switch, in: IEEE International Conference on Network Protocols (ICNP), 2018, pp. 438–439.

[512] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, K. Wehrle, Towards Executing Computer Vision Functionality on Programmable Network Devices, in: ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms, 2019, p. 15–20.

[513] J. Xie, C. Qian, D. Guo, M. Wang, S. Shi, H. Chen, Efficient Indexing Mechanism for Unstructured Data Sharing Systems in Edge Computing, in: IEEE International Conference on Computer Communications (INFOCOM), 2019, pp. 820–828.

[514] Y.-S. Lu, K. C.-J. Lin, Enabling Inference Inside Software Switches, in: Asia-Pacific Network Operations and Management Symposium (APNOMS), 2020, pp. 1–4.

[515] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, K.-K. R. Choo, P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking, Computers & Security Journal 88 (2019) 101629.

[516] T. Osiński, H. Tarasiuk, P. Chaignon, M. Kossakowski, P4rt-OVS: Programming Protocol-Independent,Runtime Extensions for Open vSwitch with P4, in: IFIP-TC6 Networking Conference (Networking), 2020, pp. 413–421.

[517] GitHub: P4rt-OVS, https://github.com/Orange-OpenSource/p4rt-ovs, accessed 01-20-2021 (2021).

[518] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, P. Richtarik, Scaling Distributed Machine Learning with In-Network Aggregation, in: USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2021.

[519] SwitchML, https://github.com/p4lang/p4app-switchML, accessed 15-02-2022 (2021).

[520] F. Yang, Z. Wang, X. Ma, G. Yuan, X. An, SwitchAgg: A Further Step Towards In-Network Computing, in: IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), 2019.

[521] S. R. Li, R. W. Yeung, N. Cai, Linear Network Coding, IEEE Transactions on Information Theory 49 (2003) 371–381.

[522] Deutsche Telekom AG: Deutsche Telekom's Access 4.0 platform goes live, https://www.telekom.com/en/media/media-information/archive/deutsche-telekom-s-access-4-0-platform-goes-live-615974, accessed 05-17-2021 (2021).

[523] O-RAN Alliance, https://www.o-ran.org/, accessed 05-17-2021 (2021).

2 Accepted Manuscripts (Additional Content)

2.1 Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)

future internet

MDPI

Article

Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)

Michael Menth [1], Habib Mostafaei [2,*,†], Daniel Merling [1] and Marco Häberle [1]

[1] Department of Computer Science, University of Tuebingen, 72076 Tubingen, Germany
[2] TU Berlin, 10623 Berlin, Germany
* Correspondence: habib@inet.tu-berlin.de
† Part of this work was conducted while this author was with Roma Tre University, Italy.

Received: 20 June 2019; Accepted: 17 July 2019; Published: 19 July 2019

Abstract: Activity-Based Congestion management (ABC) is a novel domain-based QoS mechanism providing more fairness among customers on bottleneck links. It avoids per-flow or per-customer states in the core network and is suitable for application in future 5G networks. However, ABC cannot be configured on standard devices. P4 is a novel programmable data plane specification which allows defining new headers and forwarding behavior. In this work, we implement an ABC prototype using P4 and point out challenges experienced during implementation. Experimental validation of ABC using the P4-based prototype reveals the desired fairness results.

Keywords: Activity-based congestion management (ABC); programmable data plane; QoS

---

1. Introduction

Future mobile networks like 5G consist of small cells that issue large traffic rates with high fluctuations [1]. As quality of service (QoS) is required, economic provisioning of the transport network is a challenge. Datacenter networks and residential access networks of Internet service providers (ISPs) have similar requirements [2–4].

To avoid QoS degradation, scalable bandwidth-sharing mechanisms for congestion management may be helpful, but they need to be simple and effective. That means, light users should be protected against overload caused by heavy users while avoiding per-user signaling and information within the transport network for complexity reasons.

In [5], activity-based congestion management (ABC) was initially suggested for that purpose. It implements a domain concept where edge nodes run an activity meter that measures the traffic rates of users and add activity information to their packets. Forwarding nodes leverage activity-based active queue management (activity AQM) which uses this information to preferentially drop packets from most active users in case of congestion. In [6], ABC has been proposed in its current form and extensive simulation results have demonstrated that ABC can effectively protect light users against heavy users to such an extent that a single TCP connection from a light user does not significantly suffer in the presence of congestion caused by an aggressive non-responsive traffic stream of a heavy user.

As ABC requires additional header information and new features in edge nodes and forwarding nodes, it cannot be configured on conventional networking gears. However, advances in network programmability support the definition of new headers and node behavior. The network programming language P4 is a notable example [7].

In this work, we report about a P4-based prototype for ABC. It demonstrates the technical feasibility of ABC while revealing challenges and giving hints for the enhancement of P4 support on switches. Furthermore, we present experimental results which confirm the simulative findings in [6].

The remainder of the paper is structured as follows. Section 2 briefly reviews related work. Section 3 explains the ABC concept in detail. Section 4 gives an introduction to SDN and P4. Section 5 describes the P4-based ABC implementation. Section 6 presents our evaluation methodology and reports experimental results. Finally, Section 7 concludes this work.

2. Related Work

A comprehensive overview of congestion management techniques can be found in [8]. Here, we discuss only approaches that are highly related to ABC.

Scheduling algorithms like Weighted Fair Queueing (WFQ) also manage congestion among traffic aggregates in a fair way. However, they require per-aggregate state information. In contrast, ABC requires that information only on ingress nodes of a domain, which keeps the core network simple and allows better scaling.

Core-Stateless Fair Queueing (CSFQ) [9] also improves fairness in core networks without per-flow state. Edge nodes meter the traffic rate of flows or users and record them in packet headers. Forwarding nodes leverage this information together with online rate measurement to detect congestion and to determine suitable drop probabilities for packets. In contrast to ABC, CSFQ requires more complex actions in core nodes and is less efficient [6].

Rainbow-Fair Queuing (RFQ) [10] marks packets at the edge with different colors whereby colors correspond to activities in ABC. A major difference to ABC is that packets of a flow are colored with random values from a range instead of with one specific value. The same is pursued by the fair activity meter in ABC [6], but simulations have shown that fair activity metering degrades fairness for congestion-controlled traffic. PPV [11] adopts the ideas of RFQ and is discussed as a base mechanism for "Statistical Behaviors" which are new service level specifications that are currently discussed in Metro Ethernet Forum (MEF). They differentiate the impact of congestion among different flows within a single service class while avoiding per-aggregate states in core networks. The same authors adapt PPV to deploy it in broadband access networks [12], take user activity on various time scales into account [13], and target 5G networks [14].

Fair Dynamic Priority Assignment (FDPA) [15] is a fair bandwidth sharing method for TCP-like senders which is implemented in OpenFlow and P4. Its objective is to make scheduling more scalable, but it still requires per-user state in forwarding nodes. While FDPA is applicable only for responsive traffic, ABC works with responsive traffic, non-responsive traffic, and combinations thereof.

Approximate per-flow fair-queueing (AFQ) is proposed in [16] to maintain bandwidth fairness on flow level. The authors describe how AFQ can be implemented on configurable switches in general and provide an implementation in P4. In contrast to ABC, AFQ requires per-flow state in core devices and guarantees per-flow fairness while ABC offers per-user fairness.

In [17], the authors propose CoDel (controlled delay). CoDel minimizes the time packets spend in queue by periodically checking whether the smallest sojourn time of all queued packets is below a certain threshold. If the threshold is exceeded, a packet is dropped and the time interval for the next check is decreased. As soon as the threshold is not exceeded anymore, dropping packets stops and the time interval is reset. In [18] CoDel is implemented in P4 for the software switch BMv2. Although CoDel requires floating point calculations, the authors avoid extern functions by leveraging a workaround that requires a significant amount of table entries.

In [19] the authors propose a congestion avoidance mechanism that they implement in P4. It leverages pre-established alternative paths to decrease the load on congested routes. When a network device detects that a latency-critical flow is in danger to be delayed due to queueing delay, it redirects the traffic of the affected flow to a pre-established alternative path. The authors evaluate their P4 implementation, which does not require extern functions on either the P4 software switch BMv2 or the Netronome Agilio CX SmartNIC. In contrast to ABC, the implementation utilizes a local agent on each network device to reply to congestion in a timely manner.

So far, there is only a small number of P4 implementations for congestion management mechanisms in the literature, and most of them have been demonstrated on software switches. In contrast, for monitoring purposes, there are many novel concepts based on P4 technology, and quite a few have been prototyped on hardware platforms. The work in [20] proposes a monitoring mechanism for detecting microbursts in datacenter networks at line rate. It has been implemented for the P4 hardware switch Tofino. In [21] the authors present a P4 implementation for the BMv2 software switch that monitors the network state in real-time without requiring probing packets. In a similar context the authors of [22] introduce a P4-based mechanism for the BMv2 that leverages bloom filter for both flow tracking on a single device and full path tracking. The work in [23] proposes IDEAFIX, a monitoring mechanism that identifies elephant flows in IXP networks by analyzing flow features in edge switches. IDEAFIX has been implemented for the P4 BMv2 software switch.

3. Activity-Based Congestion Management (ABC)

We give an overview of ABC, introduce the activity meter and the activity AQM in more detail, and discuss properties of ABC.

3.1. ABC Overview

ABC features a domain concept which is shown in Figure 1. Ingress nodes leverage activity meters to measure the rate of traffic aggregates that enter the ABC domain. They derive an activity value for each packet and mark that value in its header. Such an aggregate may be, e.g., the traffic of a single user or a user group. Thus, ingress nodes require traffic descriptors for any aggregate that should be tracked.

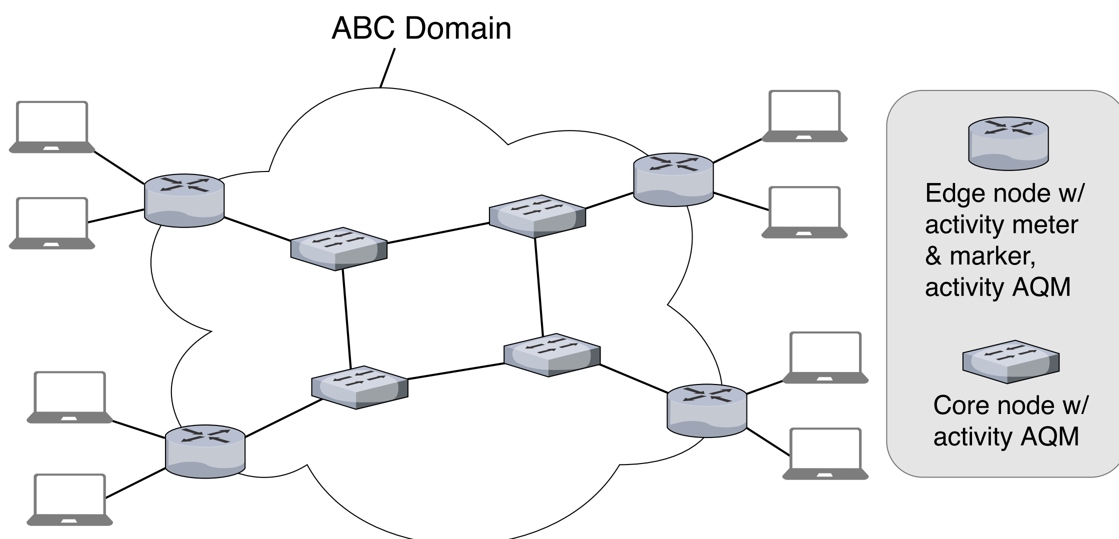


Figure 1. Activity metering and marking is performed only by ingress nodes. Both ingress and core nodes apply activity active queue management (AQM) during packet forwarding.

Ingress nodes and core nodes of an ABC domain are forwarding nodes. They use an activity AQM on each of their outgoing interfaces within the ABC domain to perform an acceptance decision for every packet. That means, they decide whether to forward or drop a packet depending on its activity. This enforces fair resource sharing among traffic aggregates within an ABC domain.

Egress nodes just remove the activity information from packets leaving the ABC domain.

3.2. Activity Meter

Ingress nodes run an activity meter per monitored traffic aggregate. The activity meter measures a time-dependent traffic rate R_m over a short time scale M_{AM} which is called memory. We chose the

TDRM-UTEMA method [24] for this purpose. The meter is configured with a reference rate R_r and computes the activity of a packet by

$$A = \log_2 \left(\frac{R_m}{R_r} \right). \quad (1)$$

The activity is written into the header of the packet before passing it to the ABC domain.

3.3. Activity AQM

An activity AQM takes acceptance decisions for packets. If the current queue size Q exceeds a computed drop threshold T_{drop}, the packet is dropped, otherwise it is forwarded. The drop threshold is computed as follows:

$$T_{drop}(A) = \max \left(Q_{min}, Q_{base} - \gamma \cdot (A - A_{avg}) \right). \quad (2)$$

We explain the components of that formula. Q_{min} prevents packet dropping in the absence of congestion. Q_{base} is a configured baseline value around which packets are dropped. A is the packet's activity and A_{avg} is a moving average of the activity of recently accepted packets. We utilize the UTEMA method [24] with memory M_{AA} for the computation of that average. The drop threshold is proportional to the packet's activity so that the loss probability of a packet increases with its activity. The parameter $\gamma > 0$ allows to tune that effect.

3.4. Discussion

Ingress nodes are configured per controlled aggregate with traffic descriptors, the memory for rate measurement, and a reference rate, and they require measurement state for each aggregate. If the number of traffic aggregates on ingress nodes is moderate, this seems feasible. Forwarding nodes are configured per egress port with parameters for activity averaging and activity AQM, and they require averaging state for each egress port. As the number of egress ports is low, ABC scales well for core nodes.

As packet dropping depends on activity values contained in packet headers, activity meters and forwarding nodes should be trusted devices. Otherwise, malicious users can avoid packet drops by inserting low activity values and obtain unfairly high throughput at the expense of other users.

Reference rates R_r specific to aggregates may be used to differentiate their achievable throughput in case of congestion. Moreover, ABC has been extended to support different delay classes, i.e., aggregate-specific throughput and forwarding delay can be controlled independently of each other. Detailed simulation results backing these claims and recommendations for parameter settings are provided in [6].

We conclude that ABC provides scalable, QoS-aware congestion management for closed networking domains.

4. Data Plane Programmability Using P4

We give an overview of data plane programmability using the programming language P4 [25] and its processing pipeline. P4 allows the definition of new header fields and forwarding behaviour, which makes it attractive for the implementation of novel forwarding paradigms. P4 programs are compiled for so-called targets, i.e., P4-capable switches, and offer a program-specific application programming interface (API) for their configuration. This API serves for either manual configuration or automatic configuration using a controller. Due to the latter, P4 is often leveraged for software-defined networking (SDN).

4.1. P4 Processing Pipeline

A P4 program defines a pipeline for packet processing which is visualized in Figure 2. It is structured into the parser, the ingress pipeline, the egress pipeline, and the deparser.

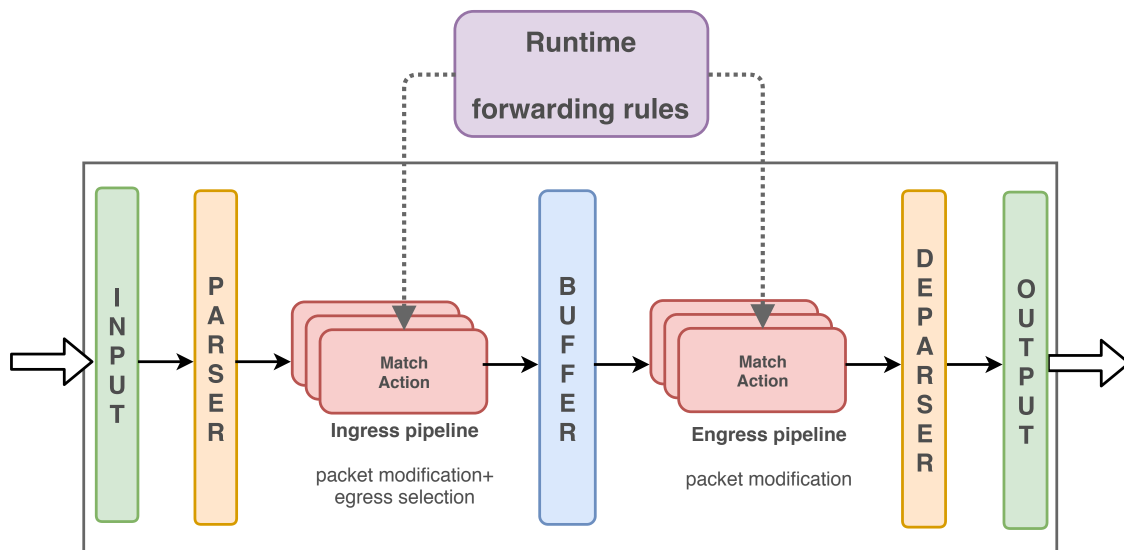


Figure 2. P4 processing pipeline.

The parser reads packet headers and stores their values in header fields. They are carried through the entire pipeline together with packets and standard metadata, e.g., the port on which the packet has been received. In addition, user-defined metadata may store values calculated during processing, e.g., flags required for decisions later in the pipeline. The ingress and egress pipeline may modify header fields, add or remove headers, clone packets, and perform many more actions useful for flexible packet processing. Packets or clones may be even processed several times by the ingress or egress pipeline, but we do not go into details here as we do not leverage these features for ABC. The ingress pipeline typically determines the output port for a packet. After completion of the egress pipeline, packets are deparsed, i.e., their headers are assembled from the possibly modified header fields, and sent.

4.2. Match Action Tables

Within the ingress and egress pipeline, packets are processed by a programmable sequence of match action tables. Their entries are called rules and each rule consists of a match part and a set of actions. Rules are installed, modified, or deleted during runtime through the API. When a packet is processed, its header or metadata fields are compared by the match part of each rule until a matching rule is found. There are different kinds of match types: exact, longest-prefix, and ternary. In case of a match, no further matching within this table is performed, and the actions in the action set of the corresponding rule are executed.

An action set consists of pre-defined primitives like adding or removing a header, reading and writing header or metadata fields, adding or subtracting values, updating counters, or dropping the packet. Custom functions, so-called externs, may be utilized within actions. Examples are encryption or decryption of fields. While the set of supported externs is target-specific, software switches even allow the definition of new externs. It is possible to define multiple match action tables. One action is applying another match action table to a packet. Thereby, packets may be processed by a chain of match action tables. To prevent processing loops, a packet can be processed at most once by any table within a pipeline.

As an example, IP forwarding can be implemented using a longest-prefix match for the destination IP address of an IP packet. In case of a match, actions are called to decrement the TTL field, adapt the IP checksum, and forward the packet to the appropriate egress port.

For the sake of readability we omit technical details about P4 programming, the P4 code, and P4 syntax. For details we refer to the $P4_{16}$ specification [25].

4.3. Variables

For arithmetic operations, P4 supports only signed integers. Therefore, we utilize extern functions for floating point operations and store floating point numbers as fixed-size bit strings. Metadata and header fields carry information throughout the processing pipeline, but they are bound to individual packets. To store information persistently, registers may be used. They can be allocated at program start and accessed and updated in actions of the ingress and egress pipeline. An example for the use of registers is keeping connection state.

5. P4-Based Implementation of ABC

We first discuss some issues that impact the overall design of the prototype. Then, we present the ingress and the egress control flow which define the behavior of the ingress and egress pipeline.

5.1. Design Considerations

We use the software switch BMv2 in the version of 10/15/2018 (unnumbered) as target. As the software switch and the transmission link are only loosely coupled, we do not have access to the buffer occupancy of the link and packets are lost in the link's buffer if the software switch sends too fast. To cope with this problem, we apply the following workaround. The BMv2 has a "packet buffer" between ingress and egress pipeline and an "output buffer" after the egress pipeline. The latter is used only for communication, not for buffering. We limit the packet rate of the egress pipeline to 4170 packets/s, which allows a throughput of 50 Mb/s for packets that are 1500 bytes large. This ensures that the transmission link cannot get overloaded and that a potential queue builds up in the BMv2's packet buffer. With ABC, packets are accepted before being buffered. Therefore, we perform packet acceptance decisions in the ingress pipeline. It requires the queue length for the egress port to which the respective packet is destined. However, this value is accessible only in the egress pipeline. Therefore, the P4 egress pipeline, whenever called to process a packet, copies the egress-port specific queue length to a register which is also accessible in the ingress pipeline. To keep that register value up-to-date, the ingress pipeline increments that register value by one whenever a new packet is accepted for a specific egress port. This design is due to the lack of BMv2 to access the buffer occupancy of the outgoing link. On hardware switches, it is desirable to have access to queue lengths of transmission links so that AQMs can be efficiently implemented.

ABC requires externs for floating point operations to support rate measurement, activity computation, activity averaging and computation of drop threshold. The externs operate on state variables. These variables are interpreted by the externs as floating point numbers but are kept as bit strings within P4. As the state variables are related to traffic aggregates or egress ports, we store their bit string values in registers. As registers cannot be read or written by externs, we copy register values to local variables and pass them to externs when they are called. Likewise, local variables can be passed to externs, modified by them, and copied back to registers after the call.

We wrote externs in C++ and manually added their code to the source code of BMv2. After recompilation, the user-defined externs were available within the P4 program. Adding externs is more difficult for hardware switches and may require vendor support.

In our experiments we study the congestion management performance of ABC on a single bottleneck link. In this particular case, only the transmitting side of the link performs activity metering and runs an activity AQM. Therefore, packets do not need to carry activity information, which removes the need for coding it in packet headers.

P4 programs provide an API for external control. We leverage this API and a script to populate match action tables and to initialize state variables in our experiments. Therefore, a controller is not needed for the prototype.

5.2. Ingress Control Flow

The ingress control flow of the ABC prototype comprises activity metering, determination of the egress port, and acceptance decision through activity AQM. It leverages two match action tables.

The first table holds rules for each aggregate with exact match on IP source address as this defines the aggregates in our experiments. The rules contain configuration parameters for activity metering and register numbers related to state variables. The associated logic calls an extern for activity metering and stores the resulting activity value in user-defined metadata that is carried with the packet. Activity metering is implemented as extern because our rate measurement requires an exponential function and floating point division, and the activity calculation further requires a logarithm (see Equation (1)). The extern leverages the packet's arrival date and size which are available as standard metadata, the configuration parameters passed as table entries, and three state variables for aggregate-specific rate measurement.

The second table determines the egress port and performs the acceptance decision. Determination of the egress port works like IP forwarding described above. Then, the egress port specific queue length is read and an extern is called that performs activity AQM including activity averaging. Besides the egress port, the rules contain configuration parameters for activity averaging and activity AQM, and register numbers related to state variables for the purpose of activity averaging. The extern accepts or denies the packet. In case of acceptance, the register for the egress port specific queue length is incremented and the packet is forwarded to the egress control flow. Otherwise, the packet is dropped.

5.3. Egress Control Flow

The egress control flow sends any received packet. In addition, it copies the egress port specific queue length to the corresponding register.

6. Performance Evaluation

We first present our evaluation methodology and then demonstrate the fairness achieved without and with ABC.

6.1. Evaluation Methodology

We describe the experimental design of our evaluation, the experimental environment, and summarize applied parameters.

6.1.1. Experimental Design

Our study quantifies the goodput achieved by two clients uploading traffic to a server over a joint bottleneck link with 50 Mb/s. Client 0 is mostly a heavy user in our experiments and Client 1 a light user. We utilize the experimental setup depicted in Figure 3. Clients with different IP addresses send traffic over fast access links with 100 Mb/s via a switch to a server behind a slow bottleneck link. Traffic from each client constitutes a traffic aggregate identified by its source IP address. In this specific experiment, only the client side of the bottleneck link requires ABC functionality. It meters the activity of packets coming from the clients and drops them depending on that value. We renounce ABC on the return path because it carries only little traffic, e.g., TCP acknowledgements in some experiment series.

6.1.2. Test Environment

Our testbed is hosted by a virtual machine with Ubuntu 16.04, 4 CPU cores with hyperthreading, 3.5 GHz, and 8 GB RAM. We utilize Mininet version 2.3.0d4 to emulate the mentioned experimental network. Clients, server, and the switch are implemented as virtual machines. We leverage Iperf 3.6 for TCP and UDP traffic generation between client and server and measure the goodput in terms of transport layer payload. One run takes 300 s and 10 runs were carried out per data point.

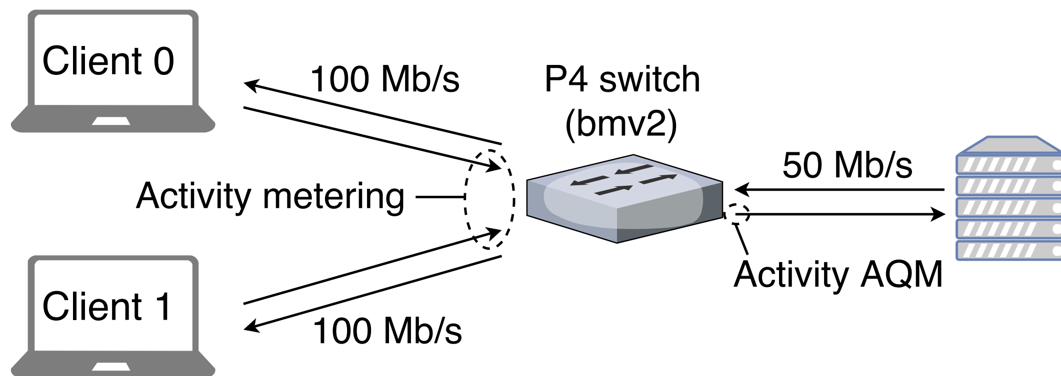


Figure 3. Experimental setup.

6.1.3. Parameters

For the sake of comparability, we choose most experimental parameters as in [6]. The clients are connected to the switch via 100 Mb/s links with a sufficiently large buffer size, and the switch is connected to the server via a 50 Mb/s link with a buffer size of 24 packets. All links are configured with a one-way delay of 5 ms.

Activity meters are configured with a memory of $M_{AM} = 3$ s and a reference rate of $R_r = 10$ kb/s. The activity averager is configured with a memory of $M_{AA} = 0.3$ s and the activity AQM utilizes the parameters $Q_{min} = 6$ packets, $Q_{base} = 20$ packets, and $\gamma = 16$ packets.

The results in this study differ from those in [6] in that we measure goodput instead of throughput, which is due to the Iperf tool, and that we utilize a larger bottleneck bandwidth of 50 Mb/s instead of 10 Mb/s in [6]. We cannot go to larger bottleneck speeds than 50 Mb/s in our experiments as the software switch BMv2 cannot read input traffic fast enough and drops packets at the ingress at higher speeds. Furthermore, we work now with $Q_{min} = 6$ packets instead of $Q_{min} = 12$ packets.

As ABC artificially limits the queue size, the transmission of a single flow may be hampered, which is undesirable. However, we validated that with the chosen parameter set; the bottleneck link can be fully utilized by a single TCP flow.

6.2. Experimental Results

We evaluate the bandwidth sharing performance without and with ABC in three different cases that were investigated by simulation in [6].

6.2.1. Resource Sharing with CBR Traffic

In a first experiment series, both clients send constant bit rate (CBR) traffic using UDP packets with 1448 bytes payload. Client 0 transmits at different rates R_0 while Client 1 sends at $R_1 = 40$ Mb/s. Figure 4 shows the obtained goodput for both clients.

Without ABC, the goodput of Client 0 continuously increases with increasing traffic rate R_0 while the goodput of Client 1 decreases. In particular, the goodput is proportional to the sent traffic rate of both clients. Thus, unfair sending behaviour is rewarded with higher goodput. This is different with ABC. The goodput of Client 0 continuously increases with increasing traffic rate R_0 as long as Client 0 sends less traffic than Client 1. As a consequence, the goodput of Client 1 continuously decreases. If Client 0 sends more traffic than Client 1, the traffic of Client 0 is preferentially dropped due to increased activity so that the goodput of Client 0 becomes clearly smaller than the goodput of Client 1. Thus, ABC creates an ecosystem in which senders benefit from decreased activity in case of congestion instead of being rewarded for sending at high rate. This incentivizes the use of congestion-controlled transport protocols.

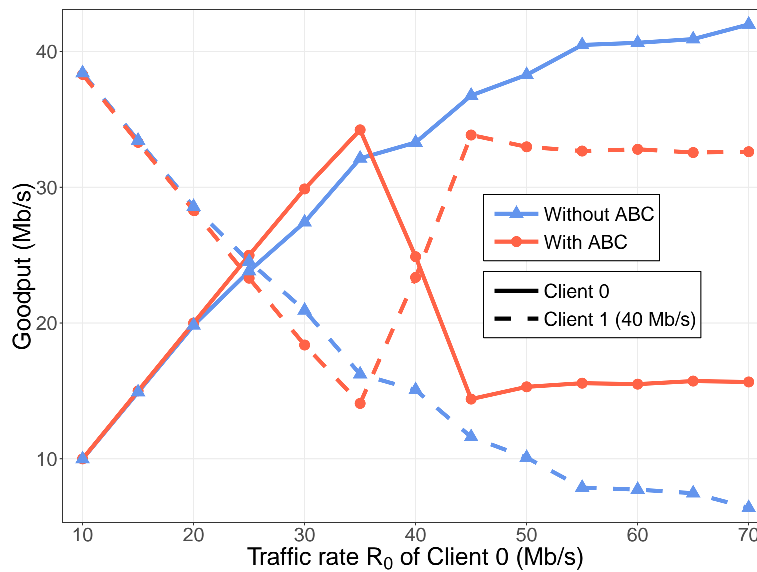


Figure 4. Resource sharing with constant bit rate (CBR) traffic; Client 0 sends CBR traffic as indicated on the *x*-axis and Client 1 sends CBR traffic at 40 Mb/s.

6.2.2. Resource Sharing with TCP Traffic

In a second experiment series, both clients send TCP traffic. We vary the number of saturated TCP connections of Client 0 while Client 1 has only a single saturated TCP connection. Thus, Client 0 is a heavy user while Client 1 is a light user. Figure 5 shows the obtained goodput for both clients.

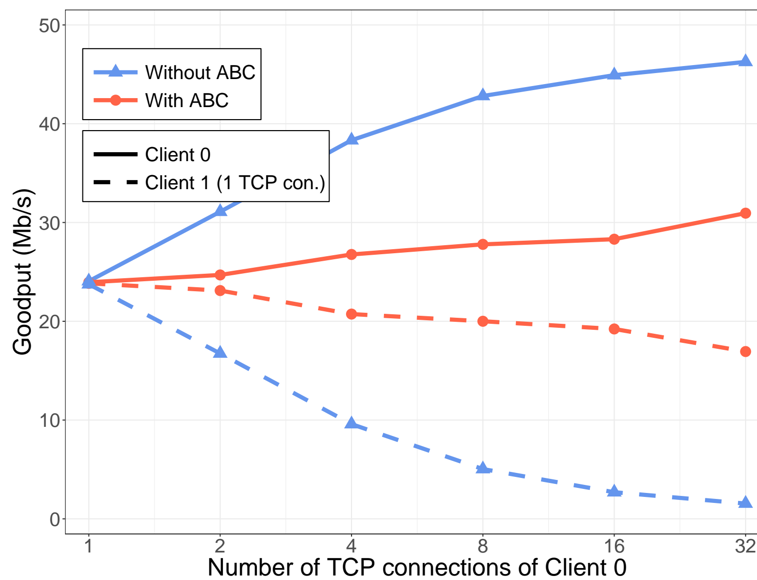


Figure 5. Resource sharing with TCP traffic; the number of saturated TCP connections of Client 0 is indicated on the *x*-axis while Client 1 has only one saturated TCP connection.

Without ABC, the goodput of Client 0 increases with increasing number of TCP connections while the goodput of Client 1 significantly decreases. With ABC, the goodput of both clients remains in the order of 25 Mb/s if the number of TCP connections for Client 0 increases. Thus, fair resource sharing is approximated.

6.2.3. Resource Sharing with CBR and TCP Traffic

In the third experiment series, Client 0 sends CBR traffic at different rates R_0 while Client 1 has a single saturated TCP connection. Figure 6 shows the obtained goodput for both clients.

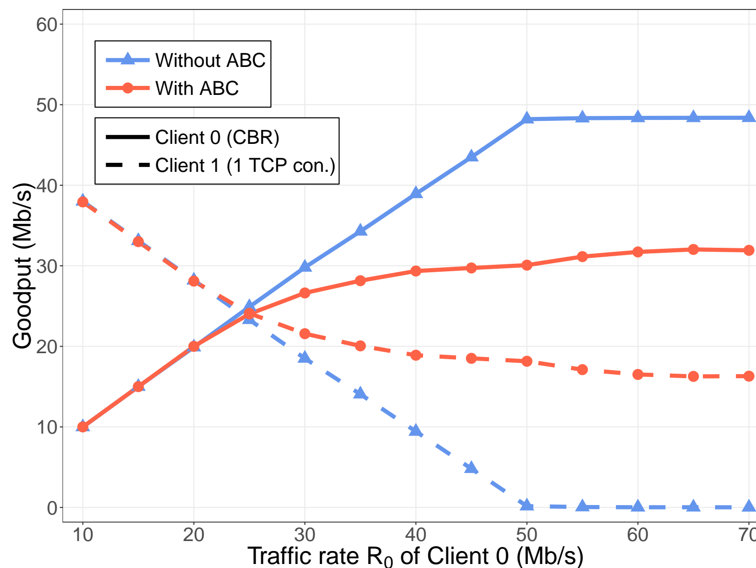


Figure 6. Resource sharing with CBR and TCP traffic; Client 0 sends CBR traffic as indicated on the *x*-axis while Client 1 has one saturated TCP connection.

Without ABC, the goodput of Client 0 increases with increasing transmission rate while the goodput of Client 1 decreases because it can only use the bandwidth left over by Client 0. If the transmission rate of Client 0 exceeds the capacity of the bottleneck link, Client 1 achieves hardly any goodput. This is different with ABC. Client 0 can increase its goodput only up to 32 Mb/s by increasing its transmission rate to very large values, but the remaining capacity is utilized by Client 1. Thus, ABC approximates fair resource sharing even under challenging conditions.

7. Conclusions

We reviewed activity-based congestion management (ABC) for fair resource sharing, gave an introduction to P4, and demonstrated the technical feasibility of ABC on programmable software switches by a prototype implementation in P4. We presented performance results illustrating the ability of ABC to enforce fairness.

The implementation leveraged several extern functions that can be utilized on a software switch but may not be available on hardware switches. Thus, P4-capable hardware switches should provide a wider range of externs to support richer use cases. Moreover, access to queue lengths of transmission interfaces are desirable to support implementation of simple AQMs.

Experimental results with the prototype implementation illustrated that ABC provides an ecosystem where users can maximize their throughput by sending at their fair share in case of congestion, which incentivizes the use of congestion controlled transport protocols. Moreover, the experimental results are in line with a more comprehensive simulation study [6].

As ABC does not require per-aggregate states in core nodes, it is a scalable technology for core networks. As it requires trusted network devices, it should be applied only in closed networks. Extensions for QoS differentiation exist. Thus, ABC provides scalable, QoS-aware congestion management for closed networking domains. Therefore, it may be an attractive technology for 5G transport networks, data center networks, or residential access networks of ISPs.

Author Contributions: Conceptualization, M.M.; Investigation, H.M.; Project administration, M.M.; Software, H.M. and M.H.; Supervision, M.M.; Validation, D.M.; Visualization, D.M. and M.H.; Writing—original draft, H.M. and D.M.; Writing—review and editing, M.M and H.M and D.M.

Funding: This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under Grant ME2727/2-1.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jaber, M.; Imran, M.A.; Tafazolli, R.; Tukmanov, A. 5G backhaul challenges and emerging research directions: A survey. *IEEE Access* **2016**, *4*, 1743–1766. [CrossRef]
2. Kutscher, D.; Mir, F.; Winter, R.; Krishnan, S.; Zhang, Y.; Bernados, C.J. Mobile Communication Congestion Exposure Scenario. 2015. Available online: http://tools.ietf.org/html/draft-ietf-conex-mobile (accessed on 1 July 2019).
3. Briscoe, B. Initial Congestion Exposure (ConEx) Deployment Examples. 2012. Available online: http://tools.ietf.org/html/draft-briscoe-conex-initial-deploy (accessed on 1 July 2019).
4. Briscoe, B.; Sridharan, M. Network Performance Isolation in Data Centres using Congestion Policing. 2014. Available online: http://tools.ietf.org/html/draft-briscoe-conex-data-centre (accessed on 1 July 2019).
5. Menth, M.; Zeitler, N. Activity-based congestion management for fair bandwidth sharing in trusted packet networks. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), Istanbul, Turkey, 23–25 April 2016.
6. Menth, M.; Zeitler, N. Fair resource sharing for stateless-core packet-switched networks with prioritization. *IEEE Access* **2018**, *6*, 42702–42720. [CrossRef]
7. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
8. Broadband Internet Technical Advisory Group (BITAG). *Real-Time Network Management of Internet Congestion*; Technical report; Broadband Internet Technical Advisory Group (BITAG): Denver, CO, USA, 2013.
9. Stoica, I.; Shenker, S.; Zhang, H. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks. *IEEE/ACM Trans. Netw.* **2003**, *11*, 33–46. [CrossRef]
10. Cao, Z.; Zegura, E.; Wang, Z. Rainbow fair queueing: Theory and applications. *Comput. Netw.* **2005**, *47*, 367–392. [CrossRef]
11. Nádas, S.; Turányi, Z.R.; Rácz, S. Per packet value: A practical concept for network resource haring. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016.
12. Laki, S.; Gombos, G.; Hudoba, P.; Nádas, S.; Kiss, Z.; Pongrácz, G.; Keszei, C. Scalable per subscriber QoS with core-stateless scheduling. *ACM SIGCOMM Demo* **2018**, *1*, 84–86.
13. Nádas, S.; Gombos, G.; Fejes, F.; Laki, S. Towards core-stateless fairness on multiple timescales. In Proceedings of the ACM/IRTF/ISOC Applied Networking Research Workshop (ANRW), Montreal, QC, Canada, 22 July 2019.
14. Nádas, S.; Turányi, Z.; Gombos, G.; Laki, S. Stateless resource sharing in networks with multi-layer virtualization. In Proceedings of the IEEE International Conference on Communications (ICC), Shangai, China, 20–24 May 2019.
15. Cascone, C.; Bonelli, N.; Bianchi, L.; Capone, A.; Sanso, B. Towards approximate fair bandwidth sharing via dynamic priority queuing. In Proceedings of the IEEE Workshop on Local & Metropolitan Area Networks (LANMAN), Osaka, Japan, 12–14 June 2017.
16. Sharma, N.K.; Liu, M.; Atreya, K.; Krishnamurthy, A. Approximating fair queueing on reconfigurable switches. In Proceedings of the USENIX Syposium on Networked Systems Design & Implementation (NSDI), Renton, WA, USA, 9–11 April 2018; pp. 1–16.
17. Nichols, K.; Jacobson, V. Controlling queue delay. *ACM Queue* **2012**, *10*, 1–15. [CrossRef]
18. Kundel, R.; Blendin, J.; Viernickel, T.; Koldehofe, B.; Steinmetz, R. P4-CoDel: Active queue management in programmable data planes. In Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, 27–29 November 2018.

19. Turkovic, B.; Kuipers, F.; van Adrichem, N.; Langendoen, K. Fast network congestion detection and avoidance using P4. In Proceedings of the ACM SIGCOMM 2018 Workshop on Networking for Emerging Applications and Technologies (NEAT), Budapest, Hungary, 20 August 2018.
20. Joshi, R.; Qu, T.; Chan, M.C.; Leong, B.; Loo, B.T. BurstRadar: Practical real-time microburst monitoring for datacenter networks. In Proceedings of the 9th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys), Jeju Island, Korea, 27–28 August 2018.
21. Geng, J.; Yan, J.; Ren, Y.; Zhang, Y. Design and implementation of network monitoring and scheduling architecture based on P4. In Proceedings of the 2nd International Conference on Computer Science and Application Engineering, Hohhot, China, 22–24 October 2018.
22. Hill, J.; Aloserij, M.; Grosso, P. Tracking network flows with P4. In Proceedings of the IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS), Dallas, TX, USA, 11 November 2018.
23. da Silva, M.V.B.; Jacobs, A.S.; Pfitscher, R.J.; Granville, L.Z. IDEAFIX: Identifying elephant flows in P4-based IXP networks. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Abu Dahbi, UAE, 9–13 December 2018.
24. Menth, M.; Hauser, F. On moving averages, histograms and time-dependent rates for online measurement. In Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE), L'Aquila, Italy, 22–27 April 2017.
25. The P4 Language Consortium. $P4_{16}$ Language Specification. Available online: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf (accessed on 1 June 2018).

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).

2.2 Load Profile Negotiation in Day-Ahead Planning for Compliance with Power Limits

Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning

Florian Heimgaertner, Sascha Heider, Thomas Stueber, Daniel Merling, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {florian.heimgaertner,thomas.stueber,daniel.merling,menth}@uni-tuebingen.de, sascha.heider@student.uni-tuebingen.de

***Abstract*—The variability of electrical energy prices at the spot market incentivizes cost-optimized load scheduling. Based on day-ahead price forecasts, energy costs can be considerably reduced by shifting energy-intensive processes to times with lower energy prices. While the mechanism of the market match demand and supply, they currently do not consider technical limitations of the electrical power grid. A large number of consumers scheduling electrical loads according to the same price forecast could result in congestion in the transmission or distribution systems.**

We propose a mechanism for day-ahead scheduling that enables negotiation of load profiles between multiple consumers and an aggregator in compliance with overall power limits. We present two mechanisms for an aggregator without knowledge about internal details of the participants to achieve this goal and compare the performance to the results of a centralized scheduler with global knowledge.

***Index Terms*—Smart grids, demand-side management, scheduling, virtual power plant.**

I. Introduction

The increasing share of weather-dependent renewable power generation leads to a large intraday variability of wholesale energy prices. Shifting loads to times with lower energy prices can considerably reduce energy costs and helps to increase the use of renewable energy by improving the match of demand and supply. Schedules of multiple consumers optimized for the same price forecast can lead to extreme load peaks. The mechanisms of the energy markets match the demand peaks and the production peaks, so the optimization of schedules based on price forecasts could be beneficial for both the generation and consumption domains. However, it can lead to problems in the transmission and distribution domains as there is no guarantee that the physical grid is capable of transporting the purchased energy volumes from generators to the consumers.

We proposed a distributed control architecture for virtual power plants [1] where participating enterprises locally optimize their load schedules according to price forecasts provided by an aggregator. The aggregator trades energy at the spot market on behalf of the participating enterprises. However, if the combined load profiles of a set of enterprises violate any constraints, the aggregator needs to negotiate re-scheduling with the affected enterprises.

In this work we propose mechanisms for a set of business units to negotiate load profiles that reduce energy costs while avoiding the violation of restrictions imposed by bottlenecks in the power grid.

This paper is structured as follows. Section II discusses related work. In Section III we present the context for the optimization and an abstract model for enterprises with load shifting capabilities. Section IV proposes two mechanisms for load profile negotiation. In Section V we shows the scenario and parameters for the evaluation and in Section VI we evaluate the performance of the negotiation mechanism and compare its results to an centralized scheduling approach with global knowledge. Section VII concludes the paper.

II. Related Work

Ibars et. al. present a distributed load management using dynamic pricing [2]. The approach is based on a network congestion game. The authors show that the system converges to a stable equilibrium. Biegel et. al. [3] describe a receding horizon control approach for moving shifting loads to minimize costs for balancing energy while avoiding grid congestion. Huang et. al. [4] propose a congestion management method for distribution grids with a high penetration of electrical vehicles and heat pumps. They use a decomposition-based optimization. In [5] they present a real-time approach for congestion management using flexible demand swap. Boroojeni et. al. [6] propose an oblivious routing economic dispatch approach for distribution grids. Bagemihl et. al. [7] describe a market-based approach to increase the capacity of a distribution grid without physical grid expansion. Hazra et. al. [8] propose a demand-response mechanism for grid congestion management using ant colony optimization. Sundström and Binding [9] propose a method for the optimization of charging schedules for electric vehicles while avoiding grid congestion.

Most work in the area of grid congestion management is based on actual grid topologies and focuses on global optimizations to avoid grid congestion. This paper uses a simplified approach, limiting congestion to a single bottleneck and focuses on interactive negotiation without global knowledge.

III. Model

In this section, we present the use case. We explain the concept of load profiles and define the parameters for the consumer model.

© 2019 VDE VERLAG GMBH

A. Use Case

The grid connection of a consumer is limited in electrical power by technical or contractual means. We denote this limit as l_c where c is a consumer. Due to limitations in the distribution grid, similar restrictions apply to groups of consumers, e.g., urban districts. As the sum of all individual power limits can be larger than the limit for the group, a group of consumers could exceed the group power limit L while still complying with their individual limits, i.e., $\sum_{c \in \mathcal{C}} l_c > L$ where $\mathcal{C}$ is a set of consumers. This problem becomes more severe in presence of price-optimized day-ahead planning when loads of all flexible consumers are scheduled for the times with the lowest energy price forecasts. However, day-ahead planning usually involves an aggregator providing the forecasts and trading at the energy markets. As an aggregator requires load forecasts of all aggregated consumers, we propose a mechanism for day-ahead demand-side management (DSM) within the group the aggregated consumers.

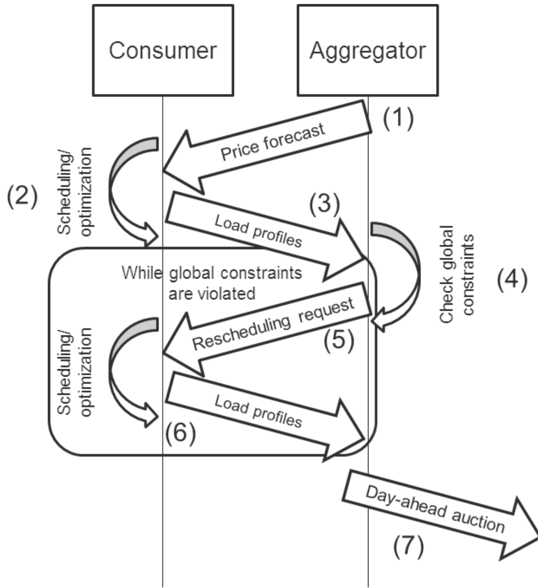


Fig. 1: Negotiation process between enterprise and aggregator during day-ahead planning.

Figure 1 shows the negotiation process to ensure that limitations for a group of consumers are complied with. The aggregator distributes price forecasts for the day-ahead energy market to the aggregated consumers (1). Each consumer computes price-optimized schedules based on their model parameters using the price forecast received from the aggregator (2). After the best schedule is selected, the consumers send the load profiles to the aggregator (3). After receiving load profiles from the consumers, the aggregator checks global constraints (4). An example for a global constraint is a cumulative power limit for a group of participants imposed by the grid operator. If such a constraint is violated, the aggregator sends a rescheduling request to the affected groups or individual participants (5). The affected consumers perform planning and optimization based on additional information provided by the aggregator and submit new load profiles (6). Steps (4)–(6) are repeated until the global constraints are no longer violated. Finally, energy is traded at the day-ahead market (7).

B. Consumer Model and Load Profiles

A load profile is a time series of electrical load over a given period. As we focus on day-ahead optimization, we chose a period of 24 hours and a granularity of one hour. A time slot is denoted as t and the set of the time slots of a day is defined as $\mathcal{T} := \{0, \dots, 23\}$. We denote the load profile of a consumer c as e_c^t, $t \in \mathcal{T}$, with an energy demand for each hour of a day. The total energy demand of all consumers in time slot t is limited by the group power limit L^t.

For our study we use an abstract model of a business consumer with flexibility for load shifting. We do not consider internal organization and dependencies among processes within a consumer, but limit the model to energy and cost parameters. The consumer is defined by a daily demand of electrical energy E_c, a power limit l_c^t, and operational costs A_c^t. The objective is to find a set of load profiles $e_c^t, t \in \mathcal{T}$, that satisfy the following conditions.

$$\sum_{t \in \mathcal{T}} e_c^t = E_c \; \forall c \in \mathcal{C} \tag{1}$$

$$e_c^t \leq l_c^t \; \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{2}$$

$$\sum_{c \in \mathcal{C}} e_c^t \leq L^t \; \forall t \in \mathcal{T} \tag{3}$$

Each load profile is associated with costs. F^t is the energy price forecast for time slot t. A_c^t gives the additional (non-energy) operation costs of a consumer c in time slot t. The total costs C_c for a consumer c are given by

$$C_c = \sum_{t \in \mathcal{T}} e_c^t \cdot F^t + A_c^t. \tag{4}$$

IV. Mechanisms

In this section, we present a linear program that computes load profiles for each participant resulting in the lowest total costs while complying with the group power limit. The linear program needs global knowledge, i.e., it requires information about internal details such as cost structures of all participants to compute the solution. However, aggregator operation without such global knowledge of internal details about the participating enterprises is an explicit goal of [1]. Therefore, we propose two methods for load profile negotiation that work without global knowledge. The sequential approval method is based on a first-come-first-serve approach combined with a compensation for swapping time slots. The simultaneous approval method requests multiple load profiles per participant to find an acceptable combination of load profiles.

A. Load Optimization Using Global Knowledge

The load profiles $e_c^t, t \in \mathcal{T}, c \in \mathcal{C}$ consist of continuous variables that can be determined by the following linear program.

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=0}^{23}\sum_{c\in\mathcal{C}} F^t e_c^t + A_c^t & \\
\text{subject to} & \sum_{c\in\mathcal{C}} e_c^t \leq L^t, & t\in\mathcal{T} \\
& \sum_{t=0}^{23} e_c^t = E_c, & c\in\mathcal{C} \\
& e_c^t \leq l_c^t, & t\in\mathcal{T}, c\in\mathcal{C} \\
& e_c^t \in \mathbb{R}, & t\in\mathcal{T}, c\in\mathcal{C}
\end{array}
$$

B. Sequential Approval of Load Profiles

For the sequential approval method, each submitted load profile is individually approved after submission unless its load combined with the previously approved load profiles would exceed the group power limit. To resolve the violation, all participants with acknowledged energy demand in the respective time intervals compute alternative load profiles avoiding the overloaded time slots $t \in \mathcal{T}'$. They submit load profiles annotated with the additional costs resulting from higher energy prices or increased operation costs in alternative time intervals. The aggregator selects the combination of load profiles with the lowest total additional costs. The process is repeated until a load profile for each participant is approved.

A linear program is used to find an appropriate combination of load profiles. The load profiles are selected such that the sum of the additional costs, i.e., the differences between the respective cheapest load profiles, of all enterprises is minimized. If every consumer c hands in n_c load profiles, let x_c^i be a binary variable which is true iff the i-th schedule of enterprise $c \in \mathcal{C}$ is selected. Furthermore, let $e_c^{t,i}$ be the energy demand of load profile i of consumer c in slot t, C_c^i the total cost of consumer c for load profile i and L^t the group power limit of slot t.

$$
\begin{array}{lll}
\text{minimize} & \sum_{c\in\mathcal{C}}\sum_{i=1}^{n_c} (C_c^i - C_c^1)\cdot x_c^i & \\
\text{subject to} & \sum_{i=1}^{n_c} x_c^i = 1, & c\in\mathcal{C} \\
& \sum_{c\in\mathcal{C}}\sum_{i=1}^{n_c} e_c^{t,i} x_c^i \leq L^t, & t\in\mathcal{T} \\
& x_c^i \in \{0,1\}, & c\in\mathcal{C}, i=1,...,n_c
\end{array}
$$

The inequations ensure that every consumer has exactly one schedule approved and that the group power limit is not exceeded in any time slot.

The participant triggering the violation compensates additional costs for participants with approved load profiles or selects a different load profile if costs are lower compared to the required compensation. While a participant can exaggerate the additional costs to generate additional revenue from rescheduling, higher costs lead to a lower chance for a load profile to be selected by the aggregator or accepted by the participant that triggers the violation.

C. Simultaneous Approval of Load Profiles

For the sequential approach the order of load profile submissions is important. Therefore late submissions of load profiles are penalized and the cost increase is distributed unevenly among the participants. This might lead to acceptance problems and prevent some enterprises from participating.

A straightforward implementation of an order-agnostic negotiation method consists of iterative energy price increases for the overloaded time slots and requests for new load profiles from all participants. However, this approach leads to artificially high energy prices and experiments showed that it fails to resolve violations for low group power limits while the sequential approval method still succeeds. Therefore, we propose a simultaneous approval method that works without modified price forecasts.

The aggregator checks for limit violations after all participants have submitted load profiles. In case of a limit violation the aggregator requests an alternative schedule from all participants, indicating the affected time slots $t \in \mathcal{T}'$. With the original load profiles and the alternative load profiles, the aggregator computes a combination not exceeding the limits. If such a combination does not exist, the aggregator repeatedly increases the number of requested load profiles per participant until there is a combination of load profiles that complies with the limits. The participants annotate the list of submitted load profiles with a preference.

The optimal selection of load profiles is computed using a linear program. If every consumer hands in n load profiles, let x_c^i be a binary variable which is true iff the i-th schedule of enterprise $c \in \mathcal{C}$ is selected. Furthermore, let $e_c^{t,i}$ be the energy demand of load profile i of consumer c in time slot t, C_c^i the total cost of consumer c for load profile i and L^t the group power limit of slot t.

$$
\begin{array}{lll}
\text{minimize} & \sum_{c\in\mathcal{C}}\sum_{i=1}^{n} i\cdot x_c^i & \\
\text{subject to} & \sum_{i=1}^{n} x_c^i = 1, & c\in\mathcal{C} \\
& \sum_{c\in\mathcal{C}}\sum_{i=1}^{n} e_c^{t,i} x_c^i \leq L^t, & t\in\mathcal{T} \\
& x_c^i \in \{0,1\}, & c\in\mathcal{C}, i=1,...,n
\end{array}
$$

The weighting of load profiles by the number i gives the load profiles a preference by the order of submission. The consumer $c \in \mathcal{C}$ indicates that a load profile $e_c^{t,i}$ is preferred over a load profile $e_c^{t,i+1}$.

V. EVALUATION MODEL

In this section we describe company-specific operational costs and day-ahead forecasts used in our experiments. Finally, we point out how load profiles are calculated for companies that participate in the negotiation processes described in Section IV-B and Section IV-C.

A. Operational Cost Factor

In the model described in Section III-B operating costs A_c^t can be given per time slot for each consumer. For our evaluation, we model the A_c^t as a dependency of the energy demand e_c^t and an operating cost factor $f_{o,c}^t$. We model the operational cost factor f^to, c of a consumer c using an interval of primary business hours and two intervals of secondary business hours. The primary business hours start at time slot t_c^p and its duration is d_c^p time slots. The secondary business hours are d_c^s time slots before and after the the primary business hours. During the primary business hours the operational cost factor is $f_{o,c}$ and $2 \cdot f_{o,c}$ during the secondary business hours. Outside of primary and secondary business hours operational costs are infinite, so business operation is not possible. The additional operational costs are given by the operational cost factor and the energy demand in the respective time slot: $A_c^t = f_{o,c}^t \cdot e_c^t$.

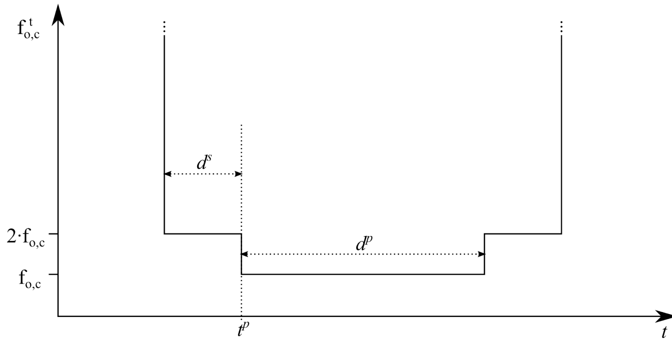


Fig. 2: Operation cost factor $f_{o,c}^t$ defined by parameters t_c^p, d_c^p, d_c^s, and n.

An example of operating costs over time defined by those parameters is given in Figure 2. The operating costs are twice as large during secondary business hours compared to primary business hours. During nonproductive hours, operating costs are infinite.

For our evaluation we chose $t_c^p \in \{7, \dots, 11\}$, $d_c^p = 8$, and $d_s^c = 2$. We define four classes of consumers by $(E_c, f_{o,c})$, $E_c \in \{1200\,\text{kWh}, 3000\,\text{kWh}\}$ and $f_{o,c} \in \{500\,€/\text{MWh}, 1000\,€/\text{MWh}\}$. The individual power limit l_c^t is set to $\frac{E_c}{6}$ in all time slots. Each starting time slot t_c^p is used once per class resulting in a group size of 20.

B. Day-Ahead Price Forecast

The prices shown in Figure 3 are used as day-ahead price forecast. While the actual prices are fictitious, the price level and the development over the 24 hour period are typical for the German day-ahead energy market.

C. Local Load Scheduling

The total costs of a schedule arise from the energy costs associated with the load profile and the operation costs. The price forecast is given as F^t, $t \in \mathcal{T}$, where F^t is the predicted price per MWh during time slot t.

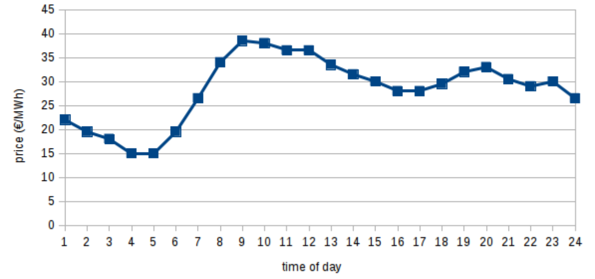


Fig. 3: Day-ahead energy price forecast.

Data: T_c^t for $0 \leq t < 24$, l_c^t, E_c
Result: e_c^t for $0 \leq t < 24$
$t[] :=$ list of times sorted ascending by value of T_c^t
$i = 0$
$E := E_c$
while $E > 0$ **do**
  if $E > l_c^t$ **then**
    $e_c^{t[i]} := l_c^t$
    $E := E - l_c^t$
  else
    $e_c^{t[i]} := E$
    $E := 0$
  $i := i + 1$
end

Algorithm 1: Cost-optimized local load scheduling.

As the hourly operation costs A_c^t in our scenario depend on the energy consumption the algorithm for producing cost-optimized schedules is straightforward. The scheduling is implemented using a greedy approach as shown in Algorithm 1. A scheduler first computes the total operation costs per kWh $T_c^t = F^t + f_{o,c}^t c$. At the time t with the lowest T_c^t, energy consumption e_c^t is set to the maximum allowed by l_c^t, proceeding with the second-lowest T_c^t and so on until $\sum_{t=0}^{23} e_c^t = E_c$. The total cost C_c of a schedule i is computed according to Equation (4).

For the computation of alternative load profiles, the consumers repeat Algorithm 1 with selectively reduced l_c^t for the affected time slots $t \in \mathcal{T}'$. For the sequential approval method, the consumers use $l_c^t = 0, \forall t \in \mathcal{T}'$. For the simultaneous approval method, the consumers reduce l_c^t for the affected time slots $t \in \mathcal{T}'$ by 1% iteratively.

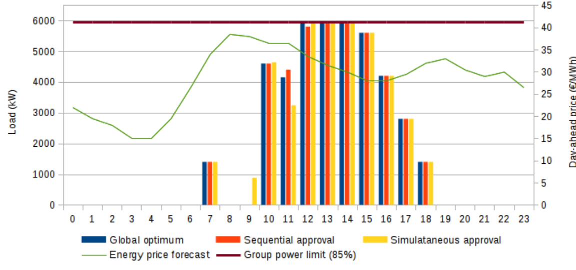
VI. Results

In this section we present the results of the evaluation. We show the load profiles resulting from sequential and simultaneous approval and compare them to the global optimum. In the evaluation scenario described in Section V, the sum of all individual power limits is given by $\sum_{c \in C} l_c^t = 7000\,\text{kW}\ \forall t \in \mathcal{T}$. We use relative group power limits of 85%, 65%, and 55%, corresponding to $L^t \in \{5950\,\text{kW}, 4550\,\text{kW}, 3850\,\text{kW}\}$ for all time slots. We show the cost increase compared to each

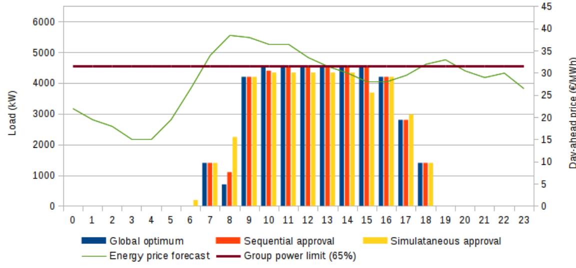
TABLE I: Relative total cost increase.

	Group power limit		
	5950 kW (85%)	4550 kW (65%)	3850 kW (55%)
Global optimum	0.03%	0.15%	4.40%
Sequential approval	0.07%	0.23%	6.01%
Simultaneous approval	0.04%	1.02%	12.12%

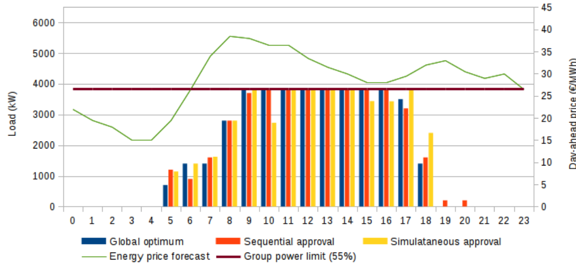
consumer's preferred load profile, which would be possible with a group power limit of $L^t = 7000\,\mathrm{kW}$. Finally we give an overview on the scheduling overhead caused by both mechanisms.



(a) Results for 85% relative group power limit.



(b) Results for 65% relative group power limit.

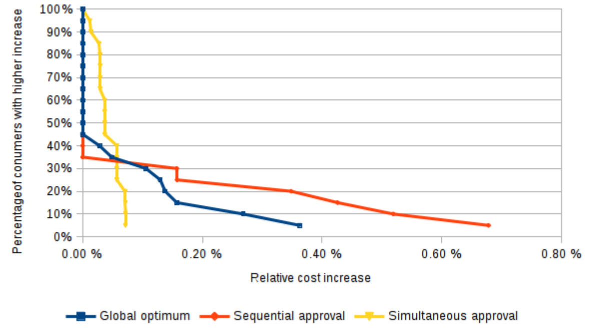


(c) Results for 55% relative group power limit.

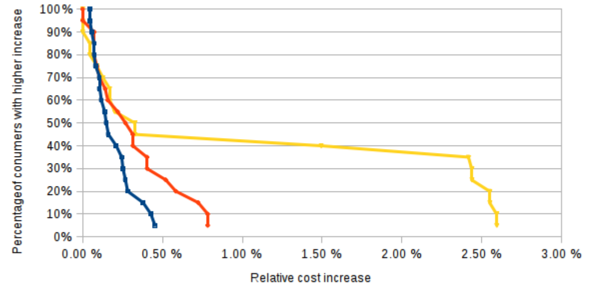
Fig. 4: Load profiles resulting from simultaneous approval, sequential approval, and global optimization at different group power limits.

A. Negotiation Results at 85% Relative Group Power Limit

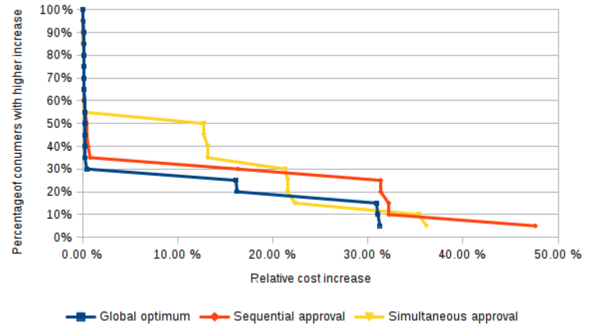
The results for the load profile negotiation at a group power limit of 5950 kW are shown in Figure 4(a). Both the sequential and simultaneous approval methods yield load profiles similar to the global optimum. The only major difference can be seen at the 9:00 time slot which is only selected in the simultaneous approval method. However, Table I shows only minimal differences regarding the increased costs. While the difference is negligible, the simultaneous approval method actually leads



(a) Results for 85% relative group power limit.



(b) Results for 65% relative group power limit.



(c) Results for 55% relative group power limit.

Fig. 5: Percentage of consumers with higher relative cost increase at different group power limits.

to lower increased costs compared to the sequential approval method. Figure 5(a) shows that no cost increase occurs for more than 50% of the consumers with the global optimum and the parallel approval method. With the simultaneous approval method, cost increase occurs for all consumers, while no consumer suffers from cost increase of more than 0.1%.

B. Negotiation Results at 65% Relative Group Power Limit

Figure 4(b) shows the results for the load profile negotiation at a group power limit of 4550 kW. While in most time slots the load is similar to the global optimum, larger differences can be seen at 6:00, 8:00, and 15:00. The sequential approval yields increased costs close to the global optimum as shown in Table I. While the increased costs caused by the simultaneous

approval method exceed the optimum by a factor of 7, with approximately 1% they are still very low. However, according to Figure 5(b) the simultaneous approval method does not only lead to the highest cost increase but also to the most uneven distribution of the cost increase among the consumers.

C. Negotiation Results at 55% Relative Group Power Limit

The results for the load profile negotiation at a group power limit of 3850 kW are shown in Figure 4(c). The low group power limit compared to the total energy demand forces the consumers to shift more energy demand to the secondary business hours. Due to the additional costs, this leads to higher total costs. In Table I we can see that even the global optimum leads to an increase of approximately 4% compared to the preferred load profile of each consumer. The sequential approval method leads to an increase of 6%, and the simultaneous approval leads to an increase of approximately 12%. Figure 5(c) does not show a significant difference regarding the evenness of the distribution of the cost increase.

D. Scheduling Overhead

Table II shows the average number of load profiles that a consumer computes before the violation of the group power limit is resolved. The sequential approval method requires the computation of slightly less load profiles compared to the simultaneous approval method.

TABLE II: Average number of load scheduling cycles per consumer.

	Group power limit		
	5950 kW (85%)	4550 kW (65%)	3850 kW (55%)
Sequential approval	17	53	90
Simultaneous approval	18	63	122

VII. Conclusion

Optimized load scheduling based on day-ahead energy price forecasts may lead to demand peaks that cannot be satisfied due to grid limitations. In this paper, we proposed approaches for load profile negotiation that do not require knowledge of internal enterprise details at the aggregator. The results for the given scenario are close to the optimum computed using global knowledge. For lower group power limits compared to the sum of all individual power limits, the sequential approval method yields a lower increase of total costs compared to the simultaneous approval method.

Due to the simplified model, the results cannot be generalized. However, the results show that it is possible to use

The first-come-first-serve property of the sequential approval method leads to penalties for late submissions and can be considered unfair. However, the expectation that the simultaneous approval method leads to a more even distribution of cost increase does not hold for low group power

load profile negotiation to comply with power limits in a day-ahead price optimization scenario. The cost increase is higher compared to a central optimization using global knowledge, but except for very low group power limits (see Section VI-C) the total cost increase is quite small.

limits. Additionally, for the simultaneous approval method an incentive for submitting the requested number of different load profiles and a distance metric to quantify the degree of difference between submitted load profiles are required.

Opportunities for future research include investigations with more complex mechanisms and more elaborated consumer models.

Acknowledgment

The research leading to these results has received funding from the German Federal Ministry for Economic Affairs and Energy under the ZIM programme (Zentrales Innovationsprogramm Mittelstand), grant no. 16KN039521. The authors alone are responsible for the content of this paper.

References

[1] F. Heimgaertner, U. Ziegler, B. Thomas, and M. Menth, "A Distributed Control Architecture for a Loosely Coupled Virtual Power Plant," in *ICE/IEEE International Technology Management Conference (ICE/IEEE ITMC)*, Jun. 2018.

[2] C. Ibars, M. Navarro, and L. Giupponi, "Distributed Demand Management in Smart Grid with a Congestion Game," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2010, pp. 495–500.

[3] B. Biegel, P. Andersen, J. Stoustrup, and J. Bendtsen, "Congestion Management in a Smart Grid via Shadow Prices," in *8th Power Plant and Power System Control Symposium (PPPSC)*, Sep. 2012.

[4] S. Huang, Q. Wu, H. Zhao, and C. Li, "Distributed Optimization based Dynamic Tariff for Congestion Management in Distribution Networks," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 184–192, 2019.

[5] S. Huang and Q. Wu, "Real-Time Congestion Management in Distribution Networks by Flexible Demand Swap," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, 2018.

[6] K. G. Boroojeni, M. H. Amini, S. S. Iyengar, M. Rahmani, and P. M. Pardalos, "An Economic Dispatch Algorithm for Congestion Management of Smart Power Networks," *Energy Systems*, vol. 8, no. 3, pp. 643–667, 2017.

[7] J. Bagemihl, F. Boesner, J. Riesinger, M. Künzli, G. Wilke, G. Binder, H. Wache, D. Laager, J. Breit, M. Wurzinger, J. Zapata, S. Ulli-Beer, V. Layec, T. Stadler, and F. Stabauer, "A Market-Based Smart Grid Approach to Increasing Power Grid Capacity Without Physical Grid Expansion," *Computer Science - Research and Development*, vol. 33, no. 1, pp. 177–183, 2018.

[8] J. Hazra, K. Das, and D. P. Seetharam, "Smart Grid Congestion Management Through Demand Response," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2012, pp. 109–114.

[9] O. Sundstrom and C. Binding, "Flexible Charging Optimization for Electric Vehicles Considering Distribution Grid Constraints," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 26–37, 2012.